Standardisierung der Kommunikation als Integrationsansatz für das Bauwesen

Dissertation

zur Erlangung des akademischen Grades

Doktor-Ingenieur

an der Fakultät Bauingenieurwesen

der

Bauhaus-Universität Weimar

vorgelegt von

Name: Ulrich Schneider

aus: Borna

Weimar

Gutachter: 1. Prof. Dr.-Ing. K. Beucke

2. Prof.Dr.Dr.h.c.mult. P.- J. Pahl

3. Prof. Dr. C. Wüthrich

Tag der Disputation: 06.Juli.2000

Inhaltsverzeichnis

1 Einleitung

1.1 Überblick

Die Verbesserung der Informations- und Kommunikationstechnologie (IuK) im Bauwesen ist ein Verlangen, welches unter anderem vom Bundesministerium für Bildung und Forschung gefordert wird. In [BMBF] heißt es: "Die Informations- und Kommunikationstechnologie ist für die Bauwirtschaft ein besonders wichtiges Querschnittsthema" und an einer anderen Stelle: "Die IuK ist die Voraussetzung für eine zügige und kostensparende Zusammenarbeit im Verbund mit vielen anderen Partnern".

Heutiges Ziel der IuK ist es, die Planungsqualität zu verbessern. Die Integration der verschiedenen Planungsunterlagen im Rechner ermöglicht einen sprunghaften Anstieg der Qualität und begründet dadurch die Aktualität des Themas. Der Qualitätssprung läßt sich auf ein effizienteres Arbeiten des Anwenders sowie das Vermeiden von Eingabefehlern durch die Übernahme von Daten zurückführen. Demnach beruht der Integrationsgedanke auf der möglichen Anwendung von Softwarelösungen für die:

- Übernahme von Daten.
- Überwachung von Veränderungen, um Fehlinformationen in den Planungsunterlagen zu vermeiden, die durch das Vergessen oder durch das unvollständige Einarbeiten durchgeführter Veränderungen zwangsweise auftreten.
- Schnelle und einfachere Ermittlung der gewünschten Informationen für die Realisierung der beiden obigen Integrationsanwendungen.

Eine Erhöhung der Planungsqualität bewirkt eine Einsparung der Kosten für die Erstellung, den Umbau sowie die Modernisierung von Bauwerken.

In den letzen Jahren betrieben Softwarehersteller und Planungsbüros erheblichen Aufwand für die Verbesserung der Kommunikations- und Informationstechnologie im Bauwesen. Diese Entwicklungen, wie STEP AP 225 ([Haas]) und IFC ([IAI99]), sind gekennzeichnet durch:

- Die Spezifikation von Bauteilen in einem semantischen Objektmodell und
- Passive Anwendung fest vorgegebener Instruktionsketten während des Integrationsprozesses.

Die in diesen Ansätzen zugrunde liegende Integrationsidee basiert auf dem semantischen, allen Entwicklern bekannten und somit notwendigerweise fest vorgegebenen Objektmodell, welches die Architektur gespeicherter Objektdaten darstellt. Jede Applikation kann entsprechend des gegebenen Aufbaues des Objektmodells ihre Daten exportieren oder benötigte Daten importieren (automatisch). Für diese Aktionen wurden Instruktionsketten vom Entwickler vorbereitet.

Dieses Vorgehen, den Anwender nicht in den Integrationsprozeß einzugliedern, hat bis jetzt noch nicht das gewünschte Ziel erreicht. Zum einen ergeben sich Nachteile aus den eingeschränkten Modellierungsmöglichkeiten bei der Erzeugung des Fachmodells während der Übernahme von Informationen, zum anderen können nur die Daten übernommen werden, die Bestandteil des semantischen Objektmodells sind.

Des weiteren stellt die Spezifikation eines semantischen Objektmodells einen erheblichen Aufwand an Zeit und Kosten dar, der vom Detailierungsgrad der zu spezifizierenden Bauteile abhängt. Komplexe Spezifikationen mit einem hohen Detailierungsgrad der verwendeten Objekte erhöhen die Fehleranfälligkeit des Objektmodells, weniger komplexe Spezifikationen vermindern den Informationsgehalt des Objektmodells. Trotzdem stellen diese Integrationsideen heutzutage den einzigen, realisierbaren Weg dar, verschiedene Planungsunterlagen zu integrieren. Im Idealfall erlaubt diese Art von Integration eine einfache und unkomplizierte Benutzung durch den Anwender.

Ähnliche Schlußfolgerungen wurden bereits 1993 in [Beu93] vorgestellt. Dennoch wurde die in diesem Vortrag favorisierte Methode, die Integration über software-technische Möglichkeiten zu realisieren, nicht in den aktuellen Integrationsansätzen aufgegriffen.

In dieser Arbeit wird ein Integrationskonzept vorgestellt, welches die favorisierte Methode in [Beu93] aufgreift und dessen wesentliches Merkmal die aktive Beteiligung des Anwenders ist. Die aktive Beteiligung des Anwenders verändert den Charakter des Integrationsprozesses vom automatischen in einen interaktiven Vorgang. Der Integrationsprozeß wird dadurch der traditionellen Arbeitsweise des Anwenders ohne Integrationssoftware ähnlicher.

Die aktive Beteiligung des Anwenders stellt die Grundlage für weitere Besonderheiten dar, die im folgenden kurz erläutert werden.

1. **Beschreibung der Objektstrukturen durch eine Grammatik:** Da der Anwender den Integrationsprozeß an der Benutzeroberfläche vollzieht, muß er die Interpretation der Daten mittels der Präsentation in der Benutzeroberfläche durchführen. Damit muß die Integrationssoftware nicht mehr die Semantik einzelner Daten kennen. Für den Zugriff auf ein ausgewähltes Datum muß die Software dessen ungeachtet die entsprechende Typinformation des Datums kennen. Die Beschreibung der Typinformation mittels einer Grammatik erlaubt einen flexibleren Umgang mit den Daten als die Standardisierung der Objekte.

2. **Handhabung der Objekte:** Da der Anwender die Kontrolle über den Ablauf des Integrationsprozesses besitzt und der Integrationsansatz interaktiv aufgebaut ist, müssen keine Vorschriften existieren, welche die Verarbeitung der einzelnen Informationen regeln. Die Handhabung der Objekte wird flexibler. Im Gegensatz dazu stehen die fest implementierten Algorithmen, die bei der Verwendung aktueller Integrationsansätze notwendig sind. Diese Algorithmen können nicht die individuellen Handhabungen aller möglichen Varianten berücksichtigen.

3. **Handhabung zum Aktualisieren bestimmter Planungsunterlagen:** Wird eine mehrfach benutzte Information in einer Planungsunterlage verändert, dann müssen alle Planungsunterlagen aktualisiert werden, die diese Information benutzen. Das Aktualisieren einer Planungsunterlage erfolgt in dem hier vorgestellten Ansatz interaktiv, d.h. unter aktiver Beteiligung des Anwenders. Der Anwender wird dabei von einem Nachrichtenkonzept

unterstützt, welches die Veränderungen einer Planungsunterlage erkennt und dazu alle Abhängigkeiten, die von den Veränderungen betroffen sind, ermittelt.

Der Anwender hat demnach in diesem Integrationsansatz die Aufgabe, den Integrationsprozeß zu kontrollieren und zu steuern. Dazu wählt er Aktionen mit den entsprechenden Parametern aus und aktiviert diese. Durch die individuelle Auswahl der Reihenfolge der Aktionen sowie der gewünschten Parameter kann er die bestimmten Aufgaben der Integration subjektiv lösen.

Die Integrationssoftware stellt den Anwendern eine Menge elementarer Aktionen zur Verfügung, die für die Realisierung der Aufgaben des Integrationsprozesses notwendig sind. Das sind Aktionen für funktionale (Manipulieren eines Objektzustandes) oder relationale (Projektion oder Verbund) Verknüpfungen. Eine besondere Aufgabe hat die Software bei der Bereitstellung von Aktionen für die Implementierung des Nachrichtenkonzeptes. Hierfür müssen überwiegend Aktionen für den Vergleich von Objekten und für das Bearbeiten von Objektbeziehungen bereitgestellt werden. Zu den Aufgaben der Software gehört nicht mehr die algorithmische Umsetzung des semantischen Wissens der während der Integration verwendeten Objekte.

Gegenstand der Arbeit ist es somit, folgende Aspekte vorzustellen und deren Einsatz für die Aufgaben der Integration zu untersuchen:

- Einbeziehung des Bearbeiters in den Integrationsprozeß für die semantische Interpretation visuell dargestellter Daten und für die interaktive Bearbeitung des Integrationsverlaufs.
- Benutzen einer Grammatik für die Beschreibung der Struktur von Objekten im Rechner.
- Spezifikation einer assoziativen Abfragesprache auf generischen Strukturen und Instanzen.
- Einsatz eines Nachrichtenkonzeptes für die Wahrung der Konsistenz.

Mit Hilfe eines Prototyps wurde die Realisierbarkeit der Konzepte beispielhaft nachgewiesen. Die Resultate, die mittels des Prototypen gewonnen wurden, zeigen die Möglichkeiten, die sich aus der flexiblen Handhabung der an der Integration beteiligten Objekte ergeben. Zum einen können jegliche Informationen direkt oder indirekt aus einer Planungsunterlage extrahiert werden. Die Einschränkung, daß nur die Informationen extrahiert werden können, die Elemente des Beschreibungsmodells sind, wurde aufgehoben. Zum anderen existiert keine Einschränkung in Bezug auf die Vielzahl der möglichen Abbildungsvorschriften, da sie nicht als Vorratsmenge in Form von aufrufbaren Algorithmen vorliegen, sondern erst vom Anwender kreiert werden. Das Benutzen eines Nachrichtenkonzeptes für die Fragen der Aktualisierung von Planungsunterlagen führte zu einer Verbesserung der Arbeitsweise des Anwenders.

Speziell für den Anwender ergeben sich Vorteile, die sich aus der schrittweisen Erzeugung seines Fachmodells herleiten. In einem Schritt werden nur ausgewählte Objekte einer Klasse in die Zielanwendung übernommen. Dadurch bleiben die Veränderungen des Modells in der Zielanwendung gering. Eine detaillierte Gegenüberstellung der Vor- und Nachteile des in dieser Arbeit vorgestellten Integrationsansatzes aus dem Blickwinkel des Anwenders wird in der Tabelle 1.1 dargestellt.

Vorteile	Nachteile
Kleine Veränderungen können leichter im Zielmodell auf Korrektheit überprüft werden.	Hoher Grad an Informatikwissen in Bezug auf das Strukturieren von Daten und Verhalten sowie deren semantische Interpretation an Hand von Benennungen bzw. Hilfedateien.
Individuelles Vorgehen bei der Erzeugung der Objekte.	Hoher individueller Aufwand für die Aufgaben der Integration.
Übernahme erfolgt auf der Ebene von Objekten und nicht auf der Ebene von Klassenbeschreibungen.	Zugriff auf die digitale Präsentation eines Modells bedingt die Verfügbarkeit der entsprechenden Applikation.
Beliebig viele Objekte können beliebig viele Instanzen erzeugen.	Performance
Beliebige Daten können aus ermittelten Daten durch die Anwendung von Operatoren gewonnen werden.	
Projektion der übernommenen Daten ausschließlich auf die gewünschten Daten.	
Das schrittweise Vorgehen, das unter Beteiligung des Anwenders erfolgt, führt zu einer konsistenten Abbildung zwischen dem Modell im Rechner und der Gedankenwelt des Anwenders. Damit wird sichergestellt, daß jedes Modellierungsdetail von den Anwendern verinnerlicht werden kann.	

Tabelle 1.1: Vor- und Nachteile des Integrationsansatzes aus der Sicht des Anwenders

Die Tätigkeit des Entwicklers beschränkt sich auf das Entwickeln von Software, welche:

- dem Anwender eine Menge von ausführbaren Aktionen bereitstellt, die die Semantik der bearbeiteten Daten nicht berücksichtigen,
- elementare Aktionen zum Vergleichen von Datenzuständen und zur Bearbeitung von Abhängigkeiten zur Verfügung stellt sowie die dafür benötigten Daten und Relationen abspeichert.

Dadurch nimmt die Komplexität der zu entwickelnden Software ab. Die Spezifikation der Software wird verständlicher, übersichtlicher und unkomplizierter. Dadurch kann die Spezifikation schneller und korrekter umgesetzt werden.

Das Ziel dieser Arbeit liegt in dem Aufzeigen eines Weges, bestehende Planungsunterlagen unterschiedlichster Softwarepakete des Bauwesens zu integrieren. Der Anwender nimmt in dem hier vorgestellten Ansatz eine besondere Stellung ein, da er mittels Aktionen die Daten aus verschiedenen Planungsunterlagen integriert. Günstig dabei ist, daß der Ansatz der Anforderung gerecht

wird, den Implementierungsaufwand minimal zu halten, der für die Generierung der Fähigkeit mit anderen Softwaresystemen zu kommunizieren benötigt wird, und dabei eine hohe Flexibilität bei der Bearbeitung der Applikationsobjekte sicherstellt.

Sobald eine Anwendung die im Abschnitt 3.5 (siehe Seite 67) aufgeführten Anforderungen erfüllt, kann diese ohne jeglichen Aufwand in den Integrationsmodul eingebetten und demnach auch für die Realisierung der Integrationsaufgaben verwendet werden.

Inhalt der Arbeit ist die Spezifikation der vom Anwender benötigten Aktionen für die Aufgaben des Integrationsprozesses sowie der Nachweis, daß dieser Integrationsansatz implementierbar ist, mit dem Ziel:

- die Vorteile des traditionellen Vorgehens mit den Vorteilen des heutigen Vorgehens in Bezug auf Integration zu vereinen,
- des Aufzeigens eines möglichen Weges zur Bewältigung der Integrationsaufgaben auf der Grundlage einer aktiven Mitarbeit des Anwenders,
- der Darstellung der unbedingt benötigten Basistechnologien basierend auf einer Minimierung des Aufwandes auf Seiten der Softwarehersteller.

Diese Arbeit soll nicht den Eindruck erwecken, einen Ansatz vorzustellen, der alle Probleme der Integration zur Zufriedenheit aller Beteiligten löst. Diese Arbeit diskutiert einen Integrationsansatz, der die Anwender in den Integrationsprozeß einbezieht und stellt Basiskonzepte vor, die durch das Einbeziehen des Anwenders notwendig werden.

1.2 Einordnung in aktuelle Forschungsarbeiten

Der in dieser Arbeit vorgestellte Integrationsansatz stellt hinsichtlich der Integrationsansätze, die auf STEP-AP225 und IFC basieren, das andere Extrem bezüglich der Aktivitäten des Anwenders bei der Realisierung der Integrationsaufgaben dar. Während bei den Integrationsansätzen STEP-AP225 und IFC der Anwender eine total passive Rolle bei der Realisierung der Integrationsaufgaben spielt, muß er bei dem in dieser Arbeit vorgestellten Integrationsansatz jede Aktion selbst auswählen und durchführen.

Für den praktischen Einsatz ist diese Arbeitsweise zu aufwendig. Der in dieser Arbeit vorgestellte Integrationsansatz beinhaltet die Möglichkeit, die Übernahme von Daten fast automatisch durchzuführen, indem der Anwender vorbereitete Abbildungsvorschriften (Kommandos) verwendet. Die jeweilige Abbildungsvorschrift fordert den Anwender zu der Auswahl der Objekte auf. Diese Objekte werden anschließend mittels vorbereiteter Kommandos für die Erzeugung neuer Objekte verwendet. Im Anschluß daran werden die so erzeugten Objekte im Integrationsmodul für die Instanziierung der Objekte in einem ausgewählten Dokument verwendet. Das Minimieren der Abbildungsvorschriften kann durch die Verwendung eines Standards, beispielsweise IFC oder STEP-AP225, erreicht werden. Reichen die Möglichkeiten der automatischen Anwendung des Integrationsmoduls nicht aus, so kann der Anwender immer noch eigene Kommandos spezifizieren.

Die Verwendung von bereitgestellten Aktionen, die einen automatischen Integrationsprozeß simulieren, sowie der mögliche Eingriff des Anwenders, sobald die bereitgestellten Aktionen nicht das

gewünschte Resultat erzielen, stellt nach Meinung des Verfassers die für das Bauwesen optimierte Variante dar, den Integrationsprozeß anwenderfreundlich zu gestalten. Der Integrationsansatz erweitert demnach die Mächtigkeit der Integrationssoftware gegenüber zur Zeit bestehender Integrationssoftware.

Da eine gute Übersicht der passiven Integrationsansätze in [Bret98] enthalten ist, soll dieser Abschnitt nicht noch einmal die verschiedenen Integrationskonzepte wiederholen.

Für die interaktive Gestaltung des Integrationsprozesses existiert zur Zeit wenig Literatur. Es gibt eine Reihe ähnlicher Forschungsprojekte, die eine aktive Bearbeitung von übertragbaren Informationen erlauben. Diese stellen jedoch den aktiven Aspekt nicht in den Vordergrund und schränken die aktiven Möglichkeiten der Anwender während der Integrationsprozesse auf die Bearbeitung der Objektbeschreibungen ein. Zu diesen Forschungsprojekten gehören FATIMA ([FAT97]) und MIKO ([MIKO97]).

Das Projekt 'Verteiltes Facility Management in Telekommunikationsnetzen' (FATIMA) ist ein Projekt des F&E-Programms der DeteBerkom GmbH und wurde in Zusammenarbeit mit dem Fachbereich Bauingenieurwesen und Angewandte Geowissenschaften der Technischen Universität Berlin bearbeitet. Ziel der Forschungsarbeit war die Entwicklung und Erprobung eines Modellverbundkonzeptes für die Konfiguration von Anwendungsumgebungen für Facility Management, welches Ähnlichkeiten mit der Integration von Modellen aufweist.

Schlüsselrolle für die Definition des Modellverbundes bildet in diesem Forschungsprojekt eine ausgewählte Person, der Integrator. Der Integrator verwendet verschiedene Werkzeuge, um eine 'mapping table' anzulegen für die semantisch äquivalenten Attribute zweier in verschiedenen Modellen befindliche Objektbeschreibungen. Um unterschiedliche Präsentationen gleicher Informationsinhalte ineinander überführen zu können, stehen dem Integrator mathematische Operationen und Konvertierungsfunktionen zur Verfügung.

Für die Realisierung des Datentransfers reicht dieser Ansatz auf Grund der eingeschränkten Möglichkeiten für die Manipulation von Objektbeschreibungen und -zuständen nicht aus. Die notwendigen Manipulationen lassen sich nicht ausschließlich von deren Objektbeschreibung herleiten, sondern sie sind vom aktuellen Zustand des Objektes und vom entsprechenden Anwender abhängig. Für die Modellintegrität wird ein umfangreicheres Konzept benötigt als die Überprüfung auf gleiche Attributwerte für alle in der 'mapping table' aufgeführten Attributpaare.

Das Projekt 'Management- und Informationssystem für den Komplettbau' (MIKO) wurde von verschiedenen Bereichen der Bauhaus-Universität und von kommerziellen Firmen gemeinsam bearbeitet. Ziel des Projektes war es, einen Informationsserver sowie separate lokale Applikations-Interfaces (Clients) zu spezifizieren, die miteinander kommunizieren können. Der Informationsserver soll den Informationsaustausch und ein Nachrichtenkonzept für ein Notifikation von Veränderungen beinhalten. Die lokalen Applikations-Interfaces sind auf die speziellen Anforderungen der Applikation zugeschnitten.

Für die Kommunikation zwischen dem Informationsserver und den einzelnen speziellen Applikations-Interfaces wurde die Kommunikationstechnologie CORBA (Common Object Request Broker Architektur) eingesetzt. Des weiteren implementiert der Informationsserver ein Metakonzept, um die verwendeten Objektbeschreibungen dynamisch im Informationsserver nachzubauen und anschließend davon die entsprechenden Objekte zu instanziieren. Das Forschungsprojekt behandelt ausführlich Konzepte, wie Benutzermanagement, Persistenz und die Handhabung von Objekt-

identifikationen. Zur Zeit unterstützt das Forschungsprojekt nur den Datentransfer, wobei immer nur ein vollständiges Modell in den Informationsserver abgelegt oder ein vollständiges Modell aus dem Informationsserver geladen wird.

Die Bearbeiter beider Forschungsprojekte (MIKO und FATIMA) verzichteten auch auf ein semantisches Objektmodell zugunsten einer flexiblen Handhabung und zugunsten einer unkomplizierteren Art, auf Veränderungen in den verwendeten Objektbeschreibungen reagieren zu können. Beide Forschungsprojekte erlauben auch die aktive Mitarbeit des Anwenders bei der Realisierung der Integrationsaufgaben.

In dieser Arbeit wird ein Integrationsansatz vorgestellt, der den Integrationsprozeß iterativ gestaltet. Jeder Iterationsschritt obliegt dabei der uneingeschränkten Kontrolle des Anwenders. Diese beiden Besonderheiten stellen für den Verfasser die wesentlichen Unterschiede zu bereits existierenden Integrationsansätzen dar. Die Realisierung basiert demnach auf allgemeingültigen Aktionen, die der Anwender interaktiv für die Realisierung seiner Integrationsaufgabe verwenden kann. Die Semantik der benötigten Aktionen darf dabei nicht von der Semantik der zugrundeliegenden Objekte abhängen. Die Spezifikation dieser allgemeingültigen Aktionen sowie die Erarbeitung der zugrunde liegenden Methodik beschreiben den Inhalt dieser Arbeit.

1.3 Begriff Integration

Über die Bedeutung des Wortes Integration können folgende Erläuterungen gefunden werden: Integration bedeutet Zusammenschluß zu einem Ganzen; Vervollständigung; Wiederherstellung; Einbeziehung; Eingliederung [Knau83].

In den Anfängen der Integrationsbetrachtungen wurden nur einfache primitive Elemente, wie Linien, Kreise oder Zahlen in den Applikationen bearbeitet. Die entsprechenden Datenfiles enthielten demnach nur diese primitiven Elemente. Heutzutage werden Applikationen benutzt, die für das Erbringen einer bestimmten Aufgabe konzipiert wurden. Diese Applikationen operieren auf komplexen Datenstrukturen, die sich aus bekannten und weniger komplexen Strukturen zusammensetzen und in ihrer Gesamtheit aber eine abstrakte Beschreibung eines Modellelementes wiedergeben. Um Informationsverluste während der Integration zu vermeiden, bestand das Ziel, diese Objekte in ihrer Gesamtheit zu übertragen und nicht ihre einzelnen Elemente.

Für die Übernahme der komplexen Anwendungsobjekte wurde die Struktur der Objekte und die semantische Bedeutung der Objekte mit ihren Komponenten festgelegt. Durch die Veröffentlichung dieser Definitionen sollte es jeder Anwendung möglich sein, einen applikationsübergreifenden Zugriff auf die Objekte anderer Applikationen zu generieren. Die Anzahl der notwendigen Prä- und Postprozessoren wurde durch eine Standardisierung minimiert.

Die weitere Entwicklung der Anwendersoftware, die eine immer größere Detaillierung der Objekte mitbrachte, führte zu Überlegungen, wie der Standard besser strukturiert werden kann. Das Ergebnis dieser Überlegungungen bestand in der Bildung von Partialmodellen. Jedes applikationsspezifische Partialmodell baute auf einer festen Menge von standardisierten Schemata auf. Damit wurde der Unübersichtlichkeit des Standards Einhalt geboten, aber ein neues Problem geschaffen: "Wie kann die Konsistenz zwischen den Objekten der einzelnen Partialmodelle sichergestellt werden?".

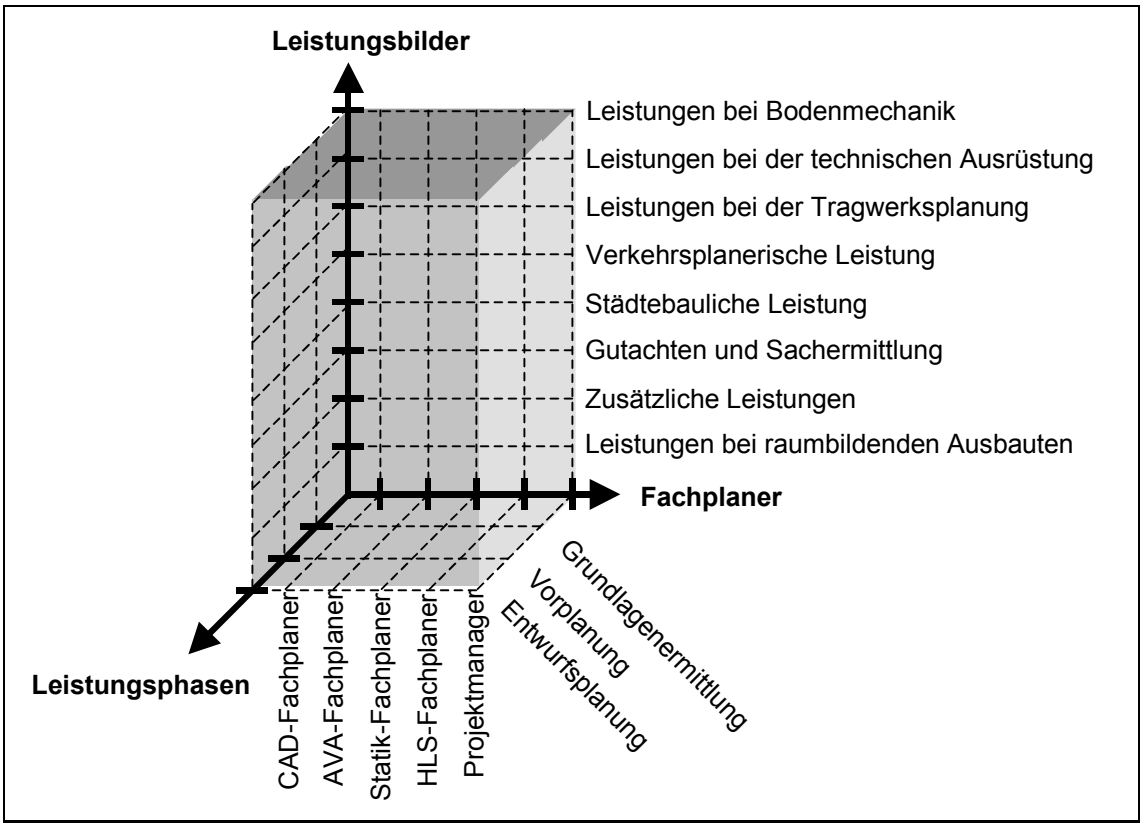


Abbildung 1.1: Klassifizierung der Modelle

Das Wort "Integration" veränderte seine semantische Bedeutung in den letzten Jahren. In den Anfängen bedeutete Integration das Exportieren sowie Importieren von primitiv strukturierten Daten, deren Datenstrukturen in Ziel- bzw. Quellenanwendung Ähnlichkeiten aufwiesen. Heute wird unter Integration die Verknüpfung von Informationen entlang der gesamten Palette von Applikationen innerhalb einer Planung verstanden, d.h. das Übernehmen der Informationen stellt nur noch einen Teil aller anfallenden Aufgaben einer Integration dar (siehe dazu folgenden Abschnitt).

1.3.1 Aufgaben der Integration

Wird der Begriff Integration in der Bausoftware benutzt, so gibt der Begriff eine allgemeine Methodik wieder, wie die Wiederverwendung von bereitgestellten Informationen realisiert werden kann. Informationen selbst werden im Rechner durch eine bestimmte Kombination von zweiwertigen Zuständen dargestellt, die mit einer Typbeschreibung verbunden sind. Die Bearbeitung der Information erfolgt durch das Starten bestimmter Aktionen innerhalb einer Applikation. Sollen Informationen wiederverwendet werden, dann müssen die Speicherzellen der Informationen in die Applikation eingebettet werden, damit die speziellen Instruktionsketten der Anwendungen die Informationen benutzen können.

Ausgangsbasis für die Betrachtung bilden demnach die heutigen Softwareprodukte. Jedes Softwareprodukt ist auf eine spezielle Modellform zugeschnitten. Die Abbildung 1.1 zeigt die möglichen Modellformen, wenn diese nach den Gesichtspunkten Leistungsphasen (Wie genau muß das Teilmodell sein?), Fachplaner (Was für ein Teilmodell ist zu erbringen?) und Leistungsbilder (Welche Modelle der physikalischen Welt sollen erstellt werden?) klassifiziert werden.

Allgemein kann gesagt werden, daß kein Softwareprodukt einen Anwender gleich komfortabel in allen

- Planungsphasen,
- Leistungsbildern und
- Fachplanungen

unterstützt. Je unterschiedlicher die zu bearbeitenden Modelle sind, um so schwieriger ist es, mit einem Softwareprodukt die gestellten Anforderungen gleich komfortabel zu realisieren. Die zur Zeit erhältlichen Softwareprodukte spezialisieren sich daher auf eine Gruppe ähnlich ineinander überführbarer Modellformen.

Die Aufgaben der Integration ergeben sich aus der Ausgangssituation und dem gestellten Ziel, die Planungsqualität zu verbessern. Die Ausgangssituation besagt, daß eine Vielzahl von unterschiedlichen Applikationen mit unterschiedlichen Modellformen betrachtet werden müssen. Das gestellte Ziel erfordert das Benutzen des Rechners für die durchzuführenden Informations- und Kommunikationsprozesse. Der Einsatz des Rechners ermöglicht zwei Maßnahmen, die direkt für eine Verbesserung der Planungsqualität stehen. Das sind:

- Vermeidung von Eingabefehlern durch die Übernahme von Informationen,
- Wahrung der Konsistenz zwischen Informationen in verschiedene Planungsunterlagen durch eine geeignete Methodik, die die Veränderung an Informationen sowie deren Abhängigkeiten protokolliert.

Vermeidung von Eingabefehlern durch die Übernahme von Informationen

Zu den Aufgaben der Integration in dieser Kategorie gehört es, eine bestimmte Information aus einem Modell (Quellenmodell) zu lesen und danach in ein anderes Modell (Zielmodell) einzufügen. Für die Aufgabe muß demnach eine Methodik bereitgestellt werden, die das Vorgehen bei der Ermittlung der gesuchten Information, bei der Überführung der ermittelten Information in das vom Zielmodell vorgeschriebene Format sowie beim Einfügen der Information beschreibt.

Die Ermittlung der gesuchten Information: Jede Applikation benutzt für die Ablage der Information Speicherzellen im Rechner. Jede Speicherzelle kann genau einen Wert aufnehmen, ein Datum, und somit eine Information speichern. Die einzelnen Speicherzellen werden mittels Strukturdefinitionen zu dem Datenraum der Applikation zusammengefaßt. Jede Applikation spezifiziert hierfür den Datenraum in einem Abstraktionsgrad, der dem zu lösenden Problem gerecht wird, und benutzt Zugriffspfade zu den Daten, die ein effizientes Arbeiten während der Anwendung der Applikation sicherstellen.

Die Ermittlung der gesuchten Information bedingt, daß für die Integrationssoftware der Zeitpunkt und der Zugriffspfad zu dem Datum, das diese Information gespeichert hat, bekannt sein muß. Dabei müssen die gegebenen, spezifischen Besonderheiten der einzelnen Applikationen herausgefunden werden, die sich aus der Beantwortung der folgenden Fragen herleiten lassen:

- Welche Daten werden von welcher Applikation zur Verfügung gestellt und welche nicht?
- Nach welchen Bearbeitungsschritten wird ein bestimmtes Datum erzeugt und somit verfügbar?
- Welche Zugriffspfade erlauben den Zugriff auf bestimmte Daten?

Überführung des Datums in das Format, welches das Zielmodell vorschreibt: Ein Modell beinhaltet eine Menge von Informationen. Einige Informationen werden in digitaler Form als Daten in dem Datenraum abgelegt, andere Informationen sind aus diesen abgelegten Daten ableitbar. Der Austausch von Informationen muß auf der Übernahme eines digitalen Wertes beruhen, der in allen beteiligten Applikation die selbe Information trägt.

Im Idealfall kann die digitale Präsentation der Information von einer Applikation in die nächste übernommen werden. Da aber jede Applikation auf einem speziellen Datenraum aufbaut, ist es zweckmäßig, ein allgemeineres Vorgehen zu unterstützen, welches die Herleitung eines Wertes aus bestehenden Daten unterstützt. Dazu müssen Abbildungsvorschriften vorgesehen werden, welche die gewünschte Darstellung der Information sowie die Berechnung des Wertes durch das Anwenden von funktionalen und relationalen Operatoren ergeben.

Funktionale Operatoren legen Abbildungsvorschriften fest, wie Werte miteinander zu verbinden sind. Mittels dieser Operatoren ist es möglich, Informationen aus bestehenden abzuleiten oder eine neue Darstellung der Information anzugeben.

Relationale Operationen legen die Beziehungen zwischen den einzelnen Atomen einer Information fest. Hiermit können neue Beziehungen definiert (relationaler Verbund), nur bestimmte Atome ausgewählt (Projektion) oder nur ausgewählte Informationsträger betrachtet (Selektion) werden. Der Einsatz dieser Operationen erlaubt die Eingrenzung auf die nur tatsächlich benötigten Informationen.

Einfügen der formatierten Information in das Zielmodell: Nachdem die Information die gewünschte Form der Zielanwendung angenommen hat, muß sie nur noch in den aktuellen Datenraum der Anwendung eingefügt werden. Dabei muß während der Integration sichergestellt werden, daß die Information, die eingefügt wird, allen Ansprüchen der Datenverwaltung genügt. Zu den Ansprüchen gehören applikationsspezifische Voraussetzungen, die für die Identifizierung, Darstellung und den Zugriff auf die Information gefordert werden. Nach dem korrekten Einfügen können die Instruktionsketten auf die digitalen Datenwerte der Information zugreifen und somit die Information bearbeiten.

Das Anbieten einer Integrationssoftware für die Aufgaben, die bei der Übernahme von Informationen auftreten, erfordert somit implementiertes Wissen über die Strukturen von den verwendeten Daten sowie Wissen über die Realisierung des Zugriffes auf diese Daten.

Wahrung der Konsistenz zwischen Informationen in verschiedene Planungsunterlagen

Wenn eine Information in mehreren Modellen existiert, muß eine Systematik von dem Integrationswerkzeug bereitgestellt werden, das sicherstellt, daß alle Eigentümer der Modelle zu einem bestimmten Zeitpunkt eine Überprüfung ausführen können, ob die verwendeten Daten, die ursprünglich die gleiche Information beinhalteten, immer noch den gleichen Informationsinhalt besitzen.

In diese Kategorien fallen Probleme, die sich mit dem Bemerken von Veränderung beschäftigen und die sich mit der Entscheidung befassen, ob eine veränderte Information mehrfach benutzt wird. Nachdem diese entsprechenden Informationen dem Anwender bereitgestellt sind, ist es seine Aufgabe, die geeigneten Maßnahmen für die Beseitigung der Inkonsistenz festzulegen. Die Aufgaben der Konsistenzwahrung werden durch die Vielzahl der verwendeten Konzepte in den einzelnen Applikationen erschwert, bedingt durch die Verwendung von verschieden Objektbeschreibungen und Zugriffspfaden für gleiche Informationen sowie unterschiedliche Abstraktionsstufen von der realen Welt.

Für beide Aufgabenkategorien spielen die beteiligten Fachmodelle die dominante Rolle. Jedes Modell kann Quelle und Ziel für die Aufgaben der Integration sein. Das Konzept für die Wahrung der Konsistenz beinhaltet somit die möglichen Verknüpfungen zwischen einem Quell- und Zielmodell sowie die Methodik von der Verwendung dieser Verknüpfungen.

1.3.2 Rahmenbedingungen für das Bauwesen

Die Rahmenbedingungen, die speziell für das Bauwesen zutreffen, lassen sich in zwei Gruppen einteilen. Zum einen sind das Rahmenbedingungen, die sich aus den organisatorischen Gegebenheiten ergeben, zum anderen Besonderheiten, die nur im Bauwesen auftreten und eine dominante Rolle spielen (vergleiche hierzu auch [Lück96]).

Für die Planung eines Bauvorhabens sind verschiedene Problemklassen zu untersuchen. Da die Bearbeitung einer Problemklasse auf ein spezielles Modellierungskonzept aufbaut, sind demnach auch verschiedene Applikationen notwendig, die jeweils ein bestimmtes Modellierungskonzept unterstützen. Die Anzahl der Applikationen, die während einer Integration zu unterstützen sind, erhöht sich noch einmal durch die Tatsache, daß für jedes Bauvorhaben eine eigene Planung erfolgt und für jede Planung die Projektmitglieder neu zusammengestellt werden, wobei jedes Projektmitglied eine bestimmte Applikation in den Planungsprozeß einbringt. Für die Integration erwächst daraus die Forderung, daß, um eine Vielzahl von sich wechselnden Applikationen integrieren zu können, kein zusätzlicher Kosten- oder Zeitaufwand auftreten darf.

Des weiteren muß die Integration auf die eingespielten Arbeitsabläufe im Bauwesen eingehen. Dazu zählt vor allem das gleichzeitige Benutzen einer Planungsunterlage von verschiedenen Bearbeitern sowie die parallele Verfeinerung bzw. Anpassung der Planungsunterlage.

Demnach muß bei der rechnergestützten Verarbeitung der Informations- und Kommunikationsprozesse im Bauwesen ein Container realisiert werden. Der Container verwaltet alle freigegebenen Planungsunterlagen, wobei er die letzte und aktuellste Planungsunterlage einer speziellen

Leistung wiederfinden muß. Des weiteren muß er den parallelen Zugriff auf eine bestimmte Planungsunterlage zu jedem Zeitpunkt unterstützen.

Kritisch ist die Beantwortung der Frage: Wo befindet sich der Container? Einerseits bedingt die rechnergestütze IuK digitale Verbindungswege, die stochastisch von den Teammitgliedern angefordert werden. Andererseits befinden sich die Teammitglieder während der Bearbeitung der Planungsunterlagen in der Regel an verschiedenen Orten. Zweckmäßig ist es, den Container an einer bekannten, von allen Teammitgliedern erreichbaren Stelle in einem Netzwerk abzulegen. Damit können die Bearbeiter zu jedem Zeitpunkt auf die aktuellsten Planungsunterlagen zurückgreifen oder einen neuen Freigabestand den anderen Bearbeitern ohne Zeitverluste zur Verfügung stellen.

Das Benutzen eines Containers, der alle angefallenden Planungsunterlagen beinhaltet, bedingt eine neue Leistung, nämlich die Administration und die Wartung des Containers. Die Honorierung dieser Leistung stellt ein Problem dar, da die HOAI (Honorarordnung für Architekten und Ingenieure) hierfür noch keine Regelung vorschreibt. Der Einsatz von Integrationssoftware verfolgt das Ziel, durch eine Verbesserung der Planungsqualität Kosten und Zeit einzusparen. Aus diesem Grund ist es zweckmäßig, den Projektplanern diese unvergütete Leistung anzutragen, da es deren Interesse ist, verschiedene Planungsunterlagen zu koordinieren.

Die Besonderheiten ergeben sich aus dem Vorgehen der Bearbeiter während eines Planungsprozesses. Während eines Planungsprozesses werden die Planungsunterlagen von den einzelnen Bearbeitern aktualisiert und in klar definierten Abständen (Freigabestand) wird die alte Planungsunterlage durch die neue ersetzt. Die Überarbeitungen werden notwendig, wenn eine Planungsunterlage verfeinert bzw. wenn ein Fehler in der Planungsunterlage beseitigt wurde oder wenn der Bauherr Veränderungen in der einmal gemachten Zielvorgabe vornimmt. Die dabei durchgeführten Veränderungen lösen eine Reihe von Aktualisierungen in Planungsunterlagen anderer Bearbeiter aus, die die Informationen aus einer veränderten Planungsunterlage verwendeten und deshalb mit einer veränderten Ausgangssituation konfrontiert werden.

Die nächste Besonderheit ergibt sich aus dem Problem, ob eine Aktualisierung, die durch eine Veränderung in einer anderen Planungsunterlage hervorgegangen ist, automatisch durchgeführt werden kann. Im Bauwesen ist eine automatische Anpassung schwierig zu realisieren. Der Ursache basiert auf der Komplexität der für die Aktualisierung notwendigen Anweisungen. Die Veränderung einer Wandbreite beispielsweise führt in einem AVA-System nicht nur zu einer neuen Mengenangabe, sondern die Wand muß einer neuen, von den Anwendern bestimmten Position zugeordnet werden. Die meisten Veränderungen benötigen bei der Aktualisierung der Planungsunterlagen die aktive Mitarbeit des Eigentümers. Die automatische Aktualisierung bringt im Bauwesen nicht den gewünschten Erfolg und läßt sich nur vereinzelt für vorhersehbare Aktualisierungen einsetzen.

Beide Besonderheiten weisen auf eine Fragestellung hin: Benötigt ein Integrationskonzept für das Bauwesen redundante Datenstrukturen, d.h. ist es notwendig, daß jede Planungsunterlage einen eigenen Datenbestand benötigt, der übernommene Daten noch einmal instanziiert? Wird diese Frage mit ja beantwortet, dann muß für das Bauwesen auf einen Integrationsprozeß zurückgegriffen werden, der auf eigenen Funktionen und Daten aufbaut. Die Funktionen und Daten des Integrationswerkzeuges werden für die Verknüpfungen zwischen den Datenräumen der einzelnen Planungsunterlagen benötigt. Diese Frage regelt somit die Architektur von Integrationssoftware.

Für die Beantwortung sind zwei Aussagen von Interesse. Zum einen ist das die Besonderheit, daß die Integration für die Aktualisierungen den alten und den neuen Zustand von Objekten benötigt, damit der Eigentümer der Planungsunterlage die Veränderungen erfassen kann. Zum anderen wird die Anwort durch die parallele Arbeitsweise der Bearbeiter bestimmt, d.h. die Bearbeiter erstellen ihre Planungsleistung auf der Basis eines Snapshots bereitgestellter Planungsunterlagen, die auch von deren Eigentümern weiter verändert werden. Faßt man beide Aussagen zusammen, so kann nur die konzeptionelle Entscheidung zweckmäßig sein, eine redundante Datenhaltung zwischen den einzelnen Planungsunterlagen zuzulassen.

Das Benutzen einer redundanten Datenhaltung zwischen den einzelnen Planungsunterlagen bringt Vorteile sowie auch Nachteile. Die Vorteile ergeben sich aus den vereinfachten Möglichkeiten, verschiedene Formate einer Information im Integrationsmodul für die Planungsunterlagen verfügbar zu halten, d.h. die Informationen können beliebig zusammengesetzt und/oder manipuliert werden. Der wesentliche Nachteil ergibt sich aus dem Aufwand, die einzelnen Planungsunterlagen konsistent zu halten. Das Aktualisieren von bestehenden Planungsunterlagen stellt hierbei das schwierigste Problem dar. Hierzu müssen Fragen beantwortet werden, wie:
Durch welche Technologie und wie soll sichergestellt werden, daß alle Leistungen im Gesamtmodell zueinander konsistent sind, wenn sie doch separat existieren?
Wie komplex ist die kleinste Einheit, die entscheidet, ob eine Einheit konsistent zu einer anderen ist?

1.4 Gliederung der Arbeit

Die Arbeit gliedert sich grundlegend in drei Teile. Ziel des ersten Teils ist es, die Integrationsvoraussetzungen vorzustellen. Der zweite Teil erläutert den Grundgedanken des Integrationsansatzes und im dritten Teil werden die verwendeten Konzepte hergeleitet sowie die Spezifikation der Realisierung angegeben und an repräsentativen Fallbeispielen verifiziert.

Der erste Teil beginnt damit, die Voraussetzungen eines Integrationskonzeptes darzustellen. Da die Modelle der einzelnen Applikationen den Dreh- und Angelpunkt bilden, beginnt das Kapitel mit einer Analyse, welche Möglichkeiten Applikationen den Anwendern bieten und welche Möglichkeiten hierfür die Entwickler einsetzen können. Im Anschluß daran wird die Rolle des Anwenders in Hinblick auf die Verwendung von Integrationssoftware oder bei Nichtverwendung beleuchtet. Die Analyse der Voraussetzungen hat das Ziel, einen formalen Überblick aller anfallenden Aktionen sowie der gegebenen Möglichkeiten von Integrationssoftware beim Integrationsprozeß wiederzugeben.

Der zweite Teil gliedert sich in zwei Abschnitte. Zunächst wird der Grundgedanke des in dieser Arbeit vorgestellten Integrationsansatzes erläutert, der maßgeblich vom gestellten Ziel abhängig ist, die Fachplaner aktiv in den Integrationsprozeß zu integrieren. Ausgehend von dieser Betrachtung erfolgt im zweiten Abschnitt die Herleitung der notwendigen Aktionen, die der Fachplaner für die Bewältigung der Integrationsaufgaben benötigt.

Der letzte Teil beginnt mit einer Auflistung von Schwerpunkten, die den Entwurf der für den Integrationsansatz benötigten Integrationssoftware maßgeblich beeinflussen und die voneinander abhängig sind. Nachdem die auf diesen Schwerpunkten basierenden Designentscheidungen erläutert sind, erfolgt die Spezifikation der Realisierung des Integrationsansatzes. Die Spezifikation wird anschließend an drei repräsentativen Beispielen für die Übernahme von Informationen

und an einem Szenario, welches die Möglichkeiten des Integrationsansatzes im Hinblick auf die Wahrung der Konsistenz diskutiert, verifiziert.

Abgerundet wird die Arbeit durch ein kurze Darstellung, welche Erweiterungen für die Anwendung des Integrationskonzeptes zweckmäßig erscheinen.

2 Beschreibung der Integrationsvoraussetzungen

Im Abschnitt 1.3.2 wurde die These aufgestellt, daß ein separater Prozeß für die Aufgaben der Integration zweckmäßig ist. Dieser Prozeß stellt eine Menge von Aktionen dem einzelnen Bearbeiter zur Verfügung, um die Daten in den aktuell bearbeiteten Modellen der einzelnen Applikation für die Aufgaben der Integration zu benutzen. Somit bilden die verwendeten Applikationen den Dreh- und Angelpunkt einer praxisgerechten Integration, die bereits im Rechner vollzogen werden soll. Für die Spezifikation der notwendigen Aktionen und der verwendeten Daten des Prozesses sind Analysen notwendig, die die Rolle der Applikationen im Integrationsprozeß und deren Umgang mit Daten beschreiben. Der Umfang für die notwendigen Aktionen ergibt sich aus der Analyse der Tätigkeiten, die ein Anwender für die Aufgaben der Integration benötigt.

2.1 Die Rolle von Applikationen im Integrationsprozeß

Applikationen sind kompilierte Anweisungen einer Programmiersprache und kodieren in Binärform die Aktionen und die Datenstrukturen der Applikationen. Die bereitgestellten Aktionen der Applikationen können in einer beliebigen, vom Anwender vorgegebenen Reihenfolge benutzt werden, um eine bestimmte, visuelle Darstellung von einem spezifischen Fachmodell zu erzeugen. Dabei erfolgt das Erzeugen des Fachmodells schrittweise, basierend auf zwei Arbeitsetappen.

Die erste Etappe beginnt mit der Auswahl einer bestimmten Aktion mit anschließender Eingabe der entsprechenden Parameter. Nach dem Aktivieren der Aktion erfolgt die Manipulation des aktuellen, von der Applikation bearbeiteten Modelles entsprechend des zugrundeliegenden Algorithmus'.

Die zweite Etappe schließt sich automatisch an die erste Etappe an. Sie erzeugt aus der modifizierten Datenstruktur die aktuelle Präsentation des Modells in der Benutzeroberfläche, wobei wiederum dafür vorgesehene Algorithmen angewendet werden. Die Algorithmen beachten für die Darstellung der Informationen die entsprechenden Normen und die anerkannten Regeln der verwendeten Symbolik des Anwenders.

Beide Etappen operieren somit auf ein und derselben aktuellen Instanz der Datenstruktur. Sie bilden somit eine Einheit zwischen dem im Computer generierten Datenmodell (Dokument) und dem dargestellten Modell (Fachmodell) in der Benutzeroberfläche. Der Umfang für die benötigten Daten und Aktionen ergibt sich aus dem Inhalt des verwendeten Modellierungskonzeptes. Das Modellierungskonzept beschreibt die verwendeten Daten und das prinzipielle Vorgehen, wie die Daten erzeugt werden und welche Manipulationen an den Daten möglich sind. Außerdem beinhaltet das Modellierungskonzept die Richtlinien für die gewünschte Darstellung einer Modellinstanz.

Die Spezifikation eines Modellierungskonzeptes stützt sich auf folgende drei Säulen:

1. **Erfahrungswerte der Anwender:** Der Anwender hat durch seine langjährige Berufserfahrung das notwendige Wissen erlangt, welche Eigenschaften (Daten) von welchen physikalischen Größen für das Erbringen seiner Leistung eine dominante Rolle spielen. Er kennt die Aktionen, die er für die Erzeugung der Daten benötigt und beherrscht die verwende-

ten Verfahren, mit denen er seine zu erbringende Leistung generiert. Im Laufe der Jahre kristallisierten sich Visualisierungsrichtlinien und Ordnungsstrukturen heraus, die eine effiziente Darstellung des Datenraums beinhalten sowie einen effizienten Zugriff auf gesuchte Informationen erlauben.

2. **Wissenschaftliche Erfahrungen und Methoden:** Dienen dem Anwender als Sammlung von Verfahren, die er für ein methodisches Herangehen bei der Lösungssuche benutzen kann. Diese Verfahren sind mit dem Stand der Werkzeuge abgeglichen, die für die Lösungssuche eingesetzt werden. Das beste Beispiel hierfür ist die Finite-Element Methode, die durch den Einsatz des Werkzeuges Rechner die statische Berechnung eines Bauwerkes revolutionierte.
3. **Normen und anerkannte Regeln:** Bezeichnen den Stand der Technik für das Erbringen einer Leistung. In den Regelwerken sind die sicheren Erfahrungswerte des Anwenders und die anerkannten wissenschaftlichen Methoden zusammengetragen.

Die Bildung eines Modellierungskonzeptes ist das Generieren einer Vorlage, die der Entwickler für die Spezifikation einer Applikation benutzt. Damit verbunden ist die Beschreibung von Bausteinen, die durch die unterschiedlichen Zusammenstellungen durch den Anwender die Modellierung verschiedener physikalischer Weltausschnitte erlauben. Die Basis des Modellierungskonzeptes ist somit das fachspezifische Modell, welches sich auf die wesentlichen Merkmale und Verfahren stützt, die für die Erbringung der Leistung benötigt werden. Die spezifischen Merkmale des Fachmodells werden durch instanziierte Daten der Bausteine beschrieben und definieren in ihrer Gesamtheit den Zustand der Modellinstanz. Die Verfahren manipulieren die Modellinstanz entsprechend den Regeln des Modellierungskonzeptes. Sie lassen sich wie folgt klassifizieren:

- das Erzeugen einer neuen Instanz eines benötigten Bausteins und das Einfügen der Instanz in die Modellinstanz,
- das Entfernen einer bestehender Instanz aus der Modellinstanz,
- das Modifizieren einer Instanz der Modellinstanz durch Nutzereingaben,
- das Aktualisieren und das Anpassen der Instanzen untereinander in einer Modellinstanz durch Berechnungsvorschriften.

Jedes spezielle Fachmodell baut auf seinem eigenen Modellierungskonzept auf. Da die heutigen Applikation nur ein Modellierungskonzept unterstützen, muß demnach ein Wechsel der Applikation durchgeführt werden, wenn sich die definierten Anforderungen an das Modellierungskonzept ändern.

2.1.1 Applikationen aus der Sicht des Anwenders

Der Anwender verfolgt mit dem Einsatz des Rechners und den darauf laufenden Applikationen folgende Ziele:

- effizienteres und genaueres Erstellen seiner Leistung (Ergebnis von durchgeführten Aktionen),

- Automatisieren von Aktionen, die auf gleichbleibenden, zeitaufwendigen und sich wiederholenden Algorithmen beruhen und

- Bereitstellen von Aktionen für die Auswertung und Überprüfung von Datenbeständen, deren Durchführung durch einen hohen Rechenaufwand gekennzeichnet ist und trotzdem mit geringen Antwortzeiten implementiert werden können.

Für die Erstellung seiner Leistung möchte der Anwender den Richtlinien des Modellierungskonzeptes folgen. Die Applikationen unterstützen den Anwender dabei durch eine Menge von Aktionen, die sich den folgenden drei Schnittstellen zuordnen lassen (siehe [Kret94] und [Hel86]):

Modelleingabeschnittstelle: Diese Schnittstelle operiert direkt auf dem zugrundeliegenden Modell der Applikation. Im wesentlichen umfaßt diese Schnittstelle Aktionen, die für die Erzeugung und Manipulation der Modellinstanz benötigt werden. Grundsätzlich lassen sich diese Aktionen den folgenden Funktionsgruppen zuordnen:

Grundfunktionen: Sind Funktionen, welche die Modellinstanz selbst nicht verändern. Sie werden aber benötigt, wenn eine Aktion die Modellinstanz verändern soll, z.B.: Identifizieren eines Objektes in der Modellinstanz, Eingabe von Positionierungsinformationen.

Manipulationsfunktionen: Sind Funktionen, welche die Modellinstanz der Applikation verändern. Diese Funktionsgruppe bietet dem Anwender die Möglichkeit, gezielt Informationen für die Generierung der spezifischen Modellinstanz von der Benutzeroberfläche in das Modell zu transferieren, z.B.: Generieren neuer Instanzen, Modifizieren von Instanzen, Löschen von Instanzen.

Hilfsfunktionen: Sind Funktionen, die dem Anwender zu jedem Zeitpunkt die Bearbeitung der Modellinstanzen erleichtern, z.B.: Eingabehilfen.

Verwaltungsfunktionen: Sind Funktionen, welche die Modellinstanz nicht verändern, aber Vorschriften zum Verwalten der Daten kennen. Dazu gehören im wesentlichen Funktionen für die persistente Ablage der Modellinstanz auf einem Speicher und Funktionen für die Kontrolle der Richtigkeit der Modellinstanz.

Ausgabeschnittstelle: Diese Schnittstelle stellt die erzeugte Modellinstanz auf der Benutzeroberfläche dar. Sie beinhaltet Funktionen für die Konfiguration der Ausgabeprimitiven und enthält die Richtlinien für die Darstellung des Modells, welche zu beachten sind.

Konfigurationsschnittstelle: Diese Schnittstelle ermöglicht eine effizientere Benutzung der Applikation. Durch das Einstellen von Optionen, Setzen von Vorgabewerten und durch das Benutzen einer eingebetteten Makro- bzw. Programmiersprache wird eine Verbesserung der Modelleingabeschnittstelle erreicht.

Der Anwender benutzt die Applikation, um Manipulationen an der Präsentation des aktuellen Modells durchzuführen. Dazu muß der Anwender Aktionen verwenden, welche von der Modelleingabeschnittstelle bereitgestellt werden. Jede Aktion wird durch die Angabe einer Benennung in der Benutzeroberfläche beschrieben. Zusätzlich kann der Anwender in den entsprechenden Handbüchern oder Online-Hilfen die semantische Bedeutung der Aktion, d.h. die Spezifikation des zugrundeliegenden Algorithmus' sowie die Beschreibung der von der Aktion verwendeten

Parameter, nachlesen. Für eine bestimmte Manipulation am Modell muß der Anwender die entsprechende Aktion aussuchen, die entsprechend dem zugrundeliegenden Algorithmus auch das gewünschte Resultat in der Benutzeroberfläche erzeugt.

Diese Ereigniskette simuliert eine Modellbildung im Rechner. Genauer gesehen wird aber kein Modell erzeugt. Die Präsentation des Fachmodells wird mittels Algorithmen aus der Datenstruktur des Dokumentes berechnet. Die Datenstruktur wird durch die startbaren Aktionen in der Benutzeroberfläche verändert, d.h. es können Daten erzeugt, verändert oder gelöscht werden.

Darin liegt auch die Tatsache begründet, daß der Anwender nur die Aktionen am Rechner durchführen kann, für die eine Anwendung auch eine Instruktionskette bereitstellt und nur die Informationen eingeben kann, für die eine Speicherzelle vorgesehen wurde.

Der Anwender hat demnach keinen Einfluß auf die zugrundeliegenden Algorithmen der Aktionen. Nachdem der Anwender eine Aktion gestartet hat, erfolgt die Abarbeitung des Algorithmus' automatisch. Der Anwender bestimmt jedoch durch die Auswahl der Aktion, durch die Angabe der Parameter und durch die zeitliche Zuordnung der Aktionen die Manipulation der Modellinstanz innerhalb der Applikation. Diese Arbeitsweise ähnelt dem traditionellen Vorgehen als noch keine Computer eingesetzt wurden. Die Arbeitsweise entspricht einem iterativen Prozeß, der nur kleine Veränderungen an der Modellinstanz pro Iterationsschritt zuläßt.

Das heißt aber auch, daß der Rechner zu keinem Zeitpunkt weder die semantische Bedeutung der Daten noch die der durchgeführten Aktionen kennt. Die Interpretation erfolgt ausschließlich durch den Entwickler und den Bearbeiter. Der Bearbeiter muß an der Benutzeroberfläche die Semantik der Aktionen interpretieren sowie die anzugebenden Parameter durch entsprechende Dateneingaben realisieren. Erleichtert wird diese Kommunikation, wenn der Entwickler das fachspezifische Vokabular des Bearbeiters in den Schnittstellen der Applikation vorsieht, d.h. die identische Übernahme gewohnter Objekte, Vorgänge und Bezeichnungen des traditionellen Modellierungskonzeptes. Damit kann die Aussage begründet werden, daß jede Applikation nur ein spezielles Modellierungskonzept unterstützt.

Die Erzeugung des Fachmodells beginnt mit der Generierung eines neuen Dokumentes. Durch weitere Aktionen können beliebige Objekte des Modellierungskonzeptes erzeugt und bearbeitet werden, bis der zu modellierende Ausschnitt der physischen Welt erzeugt wurde. Dabei ist zu beachten, daß eine Speicherzelle nur den Wert einer physikalischen Größe beinhaltet. Die physikalische Einheit des Wertes wird nicht modelliert. Der Anwender muß selbst dafür sorgen, daß alle eingegebenen Werte zueinander konsistent sind, d.h. sie müssen alle den im Modellierungskonzept vorgeschriebenen Dimensionen entsprechen.

2.1.2 Applikationen aus der Sicht des Entwicklers

Der Entwickler analysiert das Modellierungskonzept mit dem Schwerpunkt auf Datenstrukturen, Aktionen und dazugehörenden Algorithmen. Er ist dafür verantwortlich, daß das gewünschte Resultat einer ausgelösten Aktion mit der spezifizierten Reaktion in der Darstellung der Modellinstanz übereinstimmt.

Die Analyse des Modellierungskonzeptes kann methodisch erfolgen [Booch94]. Durch Gespräche mit den Anwendern oder anderen Informationsquellen wird das Modellierungskonzept in Daten-

strukturen zerlegt und die dazugehörenden Aktionen spezifiziert. Später werden die Algorithmen der Aktionen erarbeitet und die Präsentation der Daten in der Benutzeroberfläche diskutiert. Im Anschluß daran beginnt der Entwickler mit der Implementierung und mit dem Testen des Codes. Diese Phasen sind zwar logisch hintereinander angeordnet, entsprechen aber so nicht der Praxis. In der Regel ist dieser Prozeß von Rücksprüngen rekursiver Art gekennzeichnet.

Für die Implementierung einer Anwendung spielt die zu wählende Programmiersprache eine dominante Rolle. Je nach Sprache existieren unterschiedliche Werkzeuge, die speziell für die ersten Phasen eingesetzt werden. Grob lassen sich die Programmiersprachen in objektorientierte, prozedurale, transaktionsorientierte und wissensbasierte Programmiersprachen einteilen. Jede Klasse von Programmiersprachen besitzt Vor- und Nachteile für die Entwicklung eines ausgewählten, zu implementierenden Problems. Eine detaillierte Betrachtung der Möglichkeiten von Programmiersprachen kann in [Watt96], [Loe86] und [Dre89] nachgelesen werden.

Ziel des Entwicklers ist es, ein robustes Softwareprodukt zu entwickeln sowie den Anwender und demnach auch das zugrunde liegende Modellierungskonzept optimal zu unterstützen. Seine Hauptaufgabe ist somit das Entwickeln von Rechnermodellen, die eine Bearbeitung ähnlicher Problemklassen durch den Anwender erlauben. Die Daten eines Rechnermodells werden entweder vom Anwender eingegeben oder vom System generiert. Jede erzeugte Instanz eines Datums wird im Datenraum der aktuellen Modellinstanz abgelegt. Die Aktionen liegen dem Anwender in Form eines Methodenvorrates vor. Jede Methode implementiert einen spezifischen Algorithmus. Der Aufruf der Aktionen erfolgt entweder vom Anwender direkt über die Benutzeroberfläche oder ist als Reaktion indirekt mit einem ausgelösten Anwenderereignis verbunden.

Die anspruchsvollste Tätigkeit des Entwicklers ist die Abbildung des Modellierungskonzeptes in die benötigten Datenstrukturen und Algorithmen. Zusätzlich kann der Entwickler entsprechend der gewählten Programmiersprache eine unterschiedliche Granularität in der Strukturierung von Daten und Methoden erreichen.

Entwurf von Datenstrukturen

Jede Programmiersprache stellt dem Entwickler eine Menge atomarer Datentypen zur Verfügung. Atomare Datentypen können für die Ablage von Zahlen, Zeichen und Zeichenketten verwendet werden. Zusätzliche Angaben, die sich von Programmiersprache zu Programmiersprache unterscheiden, ermöglichen eine genauere Spezifikation von Wertebereich und Genauigkeit der Zahl.

Die Spezifikation eines Datums für das Rechnermodell besteht aus zwei Teilen. Der erste Teil beschreibt den Wertebereich des Datums durch die Angabe des Datentyps. Der zweite Teil wird für die Adressierung benutzt. Die Adressierung des Datums erfolgt durch einen Namen, der gleichzeitig die semantische Interpretation des Datums für den Entwickler erlaubt und eindeutig in dem gegebenen Kontext sein muß.

Mit Hilfe dieser atomaren Datentypen und durch das Verwenden von Datenkonstruktionen kann der Entwickler zusammengesetzte Datentypen erzeugen, die zusammengehörende weniger komplexe Datentypen zu einem neuen Datentyp verbinden. Traditionelle Datenkonstruktionen sind Tupel, Liste (Array) und Menge. Neben dem Wertebereich und Namen eines zusammengesetzten Typs spielen hier noch die unterstützten Zugriffe auf die Elemente des zusammengesetzten Datentyps eine dominante Rolle.

TUPEL: Der Tupelkonstrukt ermöglicht die Zusammenfassung einzelner Komponenten zu einem neuen Datentyp. Jede Komponente wird durch die Angabe des Namens mit seinem Typ beschrieben. Die einzelnen Komponententypen können unterschiedlich sein. Jeder Komponententyp muß bei der Erzeugung des Tupeltyps bekannt sein. Jede Komponente kann bei der Instanziierung des Tupeltyps einen Wert des zugrunde liegenden Wertebereichs aufnehmen. Der Zugriff erfolgt komponentenorientiert, d.h. durch die Angabe des Namens der Komponente.

LISTE, ARRAY: Diese Konstrukte ermöglichen die Zusammenfassung von mehreren Instanzen des zugrunde liegenden bekannten Datentyps zu einem neuen Typ. Die Liste kann beliebig viele Instanzen aufnehmen, das ARRAY (Feld) nur eine bestimmte Anzahl. Der Zugriff auf ein bestimmtes Element des ARRAY'S erfolgt durch die Angabe des zum Element gehörenden Indexes. Eine LISTE erlaubt die iterative Abarbeitung beginnend von der Wurzel.

MENGE: Dieser Typkonstrukt ermöglicht die Aufnahme von beliebig vielen Instanzen eines zugrunde liegenden Datentyps wie die Liste. Im Unterschied zur Liste kann zwischen den Elementen einer Menge keine Ordnung angegeben werden, welche durch die Nachfolgerbeziehung der Elemente in einer Liste (Array) definiert wird. Eine Menge kann keine Elemente doppelt enthalten. Der Zugriff auf die Elemente einer Menge erfolgt iterativ.

Neben den traditionellen zusammengesetzten Datentypen beschreibt die Algorithmenlehre noch folgende spezielle Ausprägungen der Liste.

SCHLANGE/STACK: Diese Datenstrukturen sind der LISTE ähnlich. Beide Datentypen schränken den Zugriff auf die Elemente der Datenstruktur ein. Bei der Schlange können nur neue Elemente am Ende der Liste eingefügt und am Anfang entnommen werden. Beim Stack erfolgen das Einfügen und Entnehmen eines Elementes der Liste nur am Anfang.

BAUM: Diese Datenstruktur hebt die Bedingung der Liste auf, daß jedes Listenelement nur einen Nachfolger haben kann. Ein Baumelement kann mehrere Nachfolger besitzen. Es gibt daher ein ausgezeichnetes Element, von dem aus jedes Element des Baumes erreicht werden kann.

GRAPH: Der Graph hebt noch die Voraussetzung des Baumes auf, daß jedes Element nur einen Vorgänger hat. Ein Element des Graphen (Knoten) kann mehrere Vorgänger und Nachfolger besitzen. Der Zugriff auf einen Knoten erfolgt iterativ.

Datenstrukturen dienen dazu, eine möglichst genaue Abbildung der realen Welt auch im Rechner zu simulieren. Die Struktur, mit der atomare Daten in ihrer Gesamtheit den physischen Gegenstand beschreiben sowie die Beziehungen der Daten untereinander, soll auch bei der Modellierung der Datenstrukturen im Rechner nachempfunden werden.

Die Entscheidung, welche Datenstruktur benutzt wird, ist von den auf dieser Datenstruktur operierenden Algorithmen abhängig, da Datenstrukturen maßgeblich die Effizienz der Algorithmen bestimmen und somit die Performance beeinflussen [Sed92] .

Datenstrukturen erhöhen die Übersicht darüber, welche Informationen im Rechnermodell abgelegt sind. Sie ermöglichen dem Entwickler die Beschreibung der Modelldaten auf einer höheren Abstraktionsebene und bieten somit bessere Kontrollmechanismen an, um die Konsistenz zwischen den Modellinstanzen und der resultierenden, visuellen Darstellung des Modells zu wahren.

Funktionaler Entwurf

Funktionen (Methoden) implementieren das Wissen über die angewendeten Verfahren des Modellierungskonzepts in Form von Algorithmen. Der Algorithmus greift auf den Datenraum der aktuellen Modellinstanz zu und manipuliert diesen entsprechend den implementierten Instruktionen des Algorithmus'. Andere Algorithmen stellen die Daten des Datenraums in der Benutzeroberfläche als Fachmodell dar. In ihrer Gesamtheit simulieren die Funktionen (Methoden) das entsprechende Verhalten der einzelnen Objekten aus der realen Welt im Rechner.

Ein Algorithmus besteht immer aus atomaren Bausteinen und variablen Daten, wobei die atomaren Bausteine miteinander beliebig kombinierbar sind. Atomare Bausteine sind Anweisungen für die Definition und Manipulation von Daten, für die Ein- und Ausgabe von Daten sowie Programmkontrollstrukturen, wie Verzweigung, Sequenz und Schleife. Je nach Programmiersprache werden unterschiedliche Ausprägungen unterstützt. Gekennzeichnet ist die algorithmische Abarbeitung dadurch, daß der Algorithmus Eingabedaten in Ausgabedaten umwandelt mit der Eigenschaft, daß bei gleichen Eingabedaten auch immer die gleichen Ausgabedaten berechnet werden. Die Veränderung des Zustandes von Daten stellt das Ergebnis dar, welches durch die Abarbeitung des Algorithmus' ermittelt wurde. Zusätzlich muß jeder Algorithmus nach endlicher Zeit terminieren.

Gesteuert werden die Methoden mittels Übergabeparametern. Aufgerufen werden die Methoden durch gestartete Aktionen des Anwenders in der Benutzeroberfläche oder durch andere schon aktivierte Methoden. Sie werden somit für die Kommunikation zwischen der Benutzeroberfläche und der Modelleingabeschittstelle und zur Kommunikation zwischen den Objekten innerhalb der Modellinstanz eingesetzt [Booch94].

Der Umfang der in einer Applikation verwendeten Methoden ergibt sich in erster Linie aus allen Aktionen, die der Anwender bei der Benutzung der Applikation benötigt. Hinzu kommen noch Methoden, die intern für die Kommunikation zwischen den einzelnen Daten des Modells benötigt werden.

Grundlage für die korrekte Abarbeitung der Methode ist, daß sie zu jedem Zeitpunkt die Struktur der verwendeten Daten kennt. Dieses Wissen und das Wissen, wie die Veränderungen durchzuführen sind, werden vom Entwickler der Methode in Form von Instruktionsketten bereitgestellt. Kommt es zum Aufruf der Methode, so werden die vorgedachten Instruktionen schematisch vom Rechner abgearbeit ohne jegliche Interpretation der Semantik.

Der Entwickler kann für die Spezifikation des Verhaltens der Applikation die Methoden in einer Hierarchie ordnen. Dieses Verfahren ist auch unter dem Begriff Unterprogrammtechnik bekannt.

Strukturierung von Applikationen

Neben den beiden genannten Möglichkeiten, einzelne Daten zu Datensätzen zusammenzufassen und Methoden hierarchisch anzuordnen, kann der Entwickler noch weitere Strukturierungsmöglichkeiten verwenden, um Zusammengehörendes auch in einer geschlossenen Struktur zu modellieren.

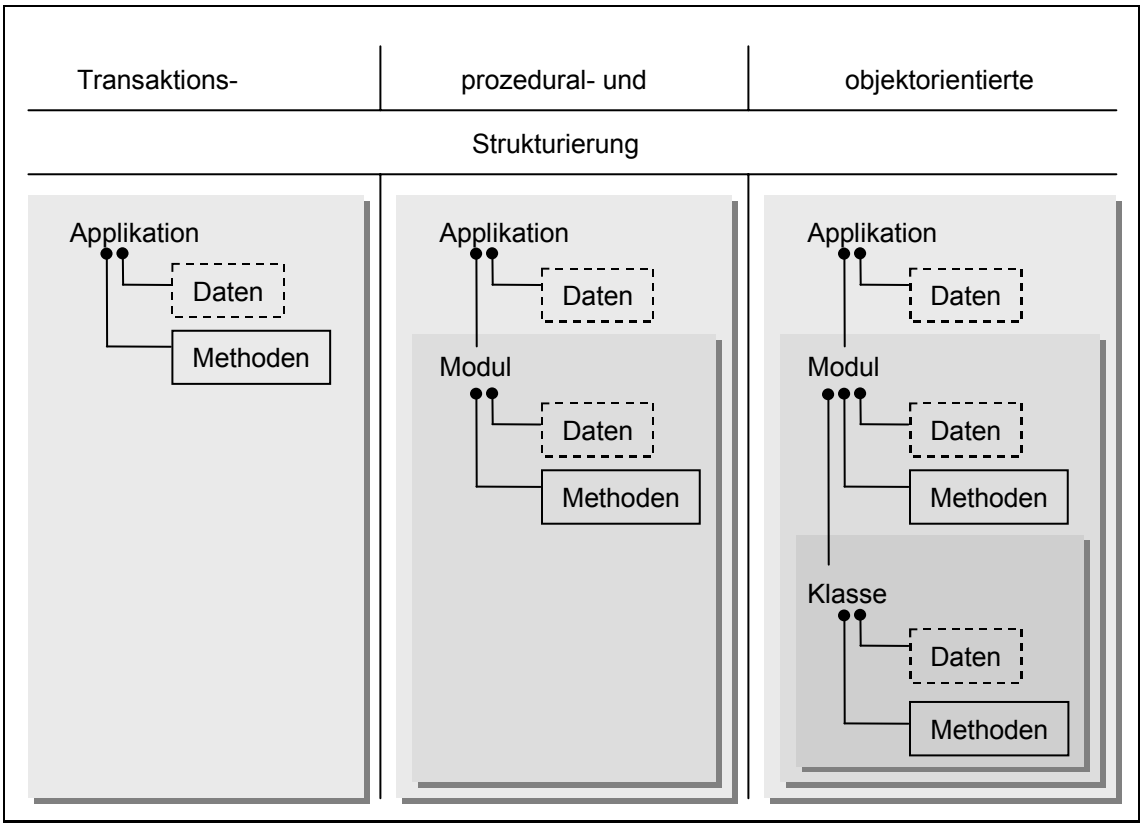


Abbildung 2.1: Strukturierung in Abhängigkeit der Programmiersprachen

Prozedurale Programmiersprachen erlauben die Modellierung von Modulen, die in sich geschlossen sind und mit den anderen Modulen über eine Modulschnittstelle (Methoden) kommunizieren. Jedes Modul kann seine eigenen, benötigten Daten und Methoden beinhalten und übernimmt eine Teilaufgabe aus der gesamten Spezifikation der Anwendung.

Objektorientierte Programmiersprachen erlauben neben der Bildung von Modulen noch die Strukturierung in Klassen. Jede Klasse kann seine eigenen Daten und Methoden definieren. Dadurch wird es erst möglich, Daten und die darauf operierenden Methoden in eine Struktur zu kapseln. Somit erlauben objektorientierte Programmiersprachen die Generierung von neuen Objekttypen, die sich, einmal eingebaut in die Programmiersprache, wie die vorhandenen Datentypen der Programmiersprache verhalten. Jede Klasse bestimmt selbst, welche Methoden für die Kommunikation mit anderen Klassen benutzt werden dürfen.

Die Abbildung 2.1 stellt noch einmal die verschiedenen Strukturierungen in Abhängigkeit der Klassifizierung von Programmiersprachen zusammen.

2.1.3 Resultierende Ausgangssituation

Der Anwender möchte mit kurzen Einarbeitungszeiten die Applikation als Werkzeug einsetzen. Er fordert von den eingesetzten Applikationen, daß sie auf die entsprechenden Objekte und

Arbeitsabläufe zugeschnitten sind und daß die Benutzeroberfläche das fachspezifische Vokabular des Anwenders unterstützt. Des weiteren will er nur die Informationen in das Modell eingeben, die er für das Erbringen seiner Leistung benötigt. Die Darstellung und Organisation der Daten muß mit den gewohnten Regeln übereinstimmen.

Der Anwender erstellt mit der Applikation die Präsentation eines Modells, die so aufbereitet ist, daß ausschließlich entsprechend geschulte Personen die semantische Deutung des Modells durchführen können. Dazu benutzt der Anwender vorbereitete Aktionen, die mit Hilfe eines Algorithmus' den Datenraum bearbeiten und anschließend die Darstellung des Fachmodells aus dem bearbeiteten Datenraum berechnen. Dabei unterliegen die eingegebenen Informationen keiner Angabe über die Dimension des spezifizierten Wertes (physikalische Größe).

Der Zustand des Modells wird ausschließlich von den eingegebenen Daten definiert. Das Verändern des Modellzustandes erfolgt durch das Auslösen von Aktionen durch den Anwender. Jede Aktion ist mit einer Folge von Instruktionen verbunden, deren implementiertes Wissen die Veränderung an den Daten in vorgegebener Weise vornimmt.

Die angewendeten Applikationen sind nicht in der Lage, eine eigene Interpretation der Semantik von Aktionen oder Daten durchzuführen. Applikationen sind Struktureinheiten, die eine bestimmte Menge von vorbereiteten Instruktionsketten zusammenfassen und diese mit einer Benutzeroberfläche verbinden. Zusätzlich verwalten sie die Daten der aktuell angelegten Modellinstanz in Form von Byteketten.

Die Implementierung der Applikation beruht auf der Definition des Datenmodells und den Methoden, welche die Modellinstanz manipulieren sowie die Berechnung der Darstellung der Modellinstanz auf der Basis des Datenmodells durchführen. Der Umfang des Datenmodells und der Methoden entspricht den verwendeten Objekten und Verfahren des Modellierungskonzeptes, das dem Fachmodell zugrunde liegt.

Die Beschreibung des Datenmodells beruht auf Festlegungen, welche Daten wie dargestellt werden. Die Darstellung des Datums basiert auf der Angabe eines Datentyps, die Adressierung des Datums beruht auf einer Benennung, die gleichzeitig die semantische Bedeutung des Datums trägt. Für die Angabe des Datentyps können atomare sowie zusammengesetzte Datentypen verwendet werden. Atomare Datentypen werden von der jeweiligen Programmiersprache zur Verfügung gestellt. Die zusammengesetzten Datentypen bestehen aus bekannten Datentypen, die mittels einer Typkonstruktion zusammengesetzt werden.

Die Beschreibung der Methoden basiert auf der Spezifikation der Übergabeparameter und der Beschreibung des Algorithmus'. Der Algorithmus definiert den Weg, wie die Eingabedaten in Ausgabedaten überführt werden. Die entsprechenden Methoden können das Datenmodell benutzen, um Daten zu instanziieren sowie bestehende Daten zu manipulieren oder zu löschen.

Je nach Wahl der Programmiersprache stehen dem Entwickler unterschiedliche Granularitäten für die Strukturierung des Datenmodells und der Methoden zur Verfügung.

2.2 Die Rolle des Anwenders im Integrationsprozeß

Die Anwender stellen neben den Applikationen die zweite externe Schnittstelle eines Integrationsansatzes dar. Aus diesem Grunde soll an dieser Stelle detailliert auf die Rolle des Anwenders eingegangen werden.

Die Rolle des Anwenders im Integrationsprozeß ist von dem Umstand abhängig, ob er Integrationssoftware benutzt oder nicht. Das Benutzen von Integrationssoftware führt zur Zeit zu einer passiven Arbeitsweise des Anwenders. Der Verzicht auf Integrationssoftware erlaubt den Anwendern aktiv die Integration von bestehenden Planungsunterlagen zu beeinflussen mit dem Nachteil einer hohen Fehleranfälligkeit. Aus diesem Grund erscheint es zweckmäßig eine Arbeitsweise herzuleiten, die den Anwender interaktiv in den Integrationsprozeß integriert, wobei die hohe Fehleranfälligkeit durch das Benutzen von Software eingeschränkt wird.

Die gewählte Arbeitweise beeinflußt nicht die Voraussetzung und das zu erwartende Ergebnis einer praktizierten Integration. Der Anwender muß am Ende gewährleisten, daß seine Planungsunterlagen mit den übernommenen Planungsunterlagen zusammenpassen. Deswegen muß der Anwender Informationen aus anderen Planungsunterlagen benutzen und seine Planungsunterlage den anderen Anwendern zur Verfügung stellen.

Durch das parallele Arbeiten der Planer treten zwangsläufig Konflikte in den Planungsunterlagen auf. Die Beseitigung der Konflikte beruht auf Kompromißlösungen, die von den betroffenen Planern erarbeitet werden.

2.2.1 Interaktive Arbeitsweise ohne Integrationssoftware

Bei der traditionellen Arbeitsweise erhält der Planer die Planungsleistung vorangegangener Phasen auf dem Medium Papier. Er muß die einzelnen Darstellungen interpretieren und seine benötigten Information auslesen, abmessen oder aus den ermittelten Informationen berechnen. Mit Hilfe einer Applikation und den Informationen, die er aus den bestehenden Planungsunterlagen gewonnen hat, erzeugt oder bearbeitet er iterativ sein entsprechendes Fachmodell. Müssen während der Bearbeitung Informationen bereitgestellt werden, die nicht in den bestehenden Planungsunterlagen enthalten sind, so wählt der Planer entsprechend seines Fachwissens geeignete Eingabedaten. Nach dem Fertigstellen seiner Planungsunterlagen erstellt er die entsprechende, durch Richtlinien und Normen vorgeschriebene Präsentation, druckt diese aus und stellt den Ausdruck anderen, darauf aufbauenden Planern zur Verfügung.

Das Erkennen von Veränderungen in den benutzten Planungsunterlagen erfolgt entweder durch das aufmerksame Verfolgen der Einträge in dem Projektbuch, welches die Art und den Umfang von Veränderungen protokolliert und von den einzelnen Planern aktualisiert wird, und durch Gespräche mit den Planern, welche die entsprechenden Planungsunterlagen erarbeiteten und demnach die Veränderungen mitteilen können, sowie durch den Vergleich der neueren Planungsunterlage mit vorhergehenden. Danach liegt es im Ermessen des Anwenders, die notwendigen Korrekturen an seiner Planungsunterlage entsprechend den gegebenen Veränderungen durchzuführen.

Die Arbeitsweise dabei entspricht der gewünschten Arbeitsweise. Interaktiv manipuliert er das Fachmodell, indem er schrittweise neue Instanzen erzeugt, bestehende Instanzen manipuliert oder löscht. Diese Arbeitsweise erlaubt dem Planer bestehende Informationen so zu verwerten, wie er sie im Kontext seines Fachmodells benötigt. Er kann während des Integrationsprozesses aktiv und individuell entscheiden, wie die gewonnenen Informationen zu verarbeiten sind. Die verwendeten Aktionen der Integration unterliegen dabei keinen Vorschriften, die die Handhabung der Informationen einschränken. Des weiteren berücksichtigt diese Arbeitsweise nur die Informationen, die der Anwender für die Realisierung seiner Planungsunterlage benötigt.

Die wesentlichen Vorteile der interaktiven Arbeitsweise liegen im flexiblen Umgang mit den Informationen und in der nachvollziehbaren Überprüfung von Veränderungen. Nachvollziehbar bedeutet, daß die iterative und interaktive Arbeitsweise des Anwenders nur kleine Veränderungen am Fachmodell erlaubt, die der Planer schnell und einfach auf deren Richtigkeit überprüfen kann.

Die Wahrscheinlichkeit, daß die Planungsunterlagen nicht zueinander passen, ist bei dieser Arbeitsweise dementsprechend hoch. Die häufigsten Fehlerquellen ergeben sich aus falschen Eingabewerten von übernommenen Informationen sowie aus dem unvollständigen Aktualisieren einer Planungsunterlage.

2.2.2 Automatische Arbeitsweise mit Integrationssoftware

Bei der automatischen Arbeitsweise verbindet der Anwender die verwendete Applikation mit einer zur Verfügung gestellten applikationsübergreifenden Datenbasis. Für diese Verbindung stellt die Applikation eine Reihe vorbereiteter Aktionen bereit, die einen bestimmten, vorgedachten Algorithmus zum Gegenstand haben. Einmal gestartet, exportiert der Algorithmus die in der Applikation enthaltenen Daten in die applikationsübergreifende Datenbasis oder erzeugt in der aktuellen Modellinstanz der Applikation Instanzen in Abhängigkeit von den vorhandenen Daten in der Datenbasis. Für fehlende Daten, die für die Erzeugung der Instanzen in der aktuellen Modellinstanz oder in der applikationsüberreifenden Datenbasis benötigt werden, werden die Standardannahmen der Werte benutzt, die bei der Erzeugung der Instanzen vorgegeben werden.

Die Wahrung der Konsistenz beruht in der automatischen Arbeitsweise auf zwei Aspekten. Zum einen existiert eine Technologie, um die Veränderung von Daten zu bemerken, zum anderen wird eine automatische Aktualisierung von Daten auf der Basis von redundanzfreien Partialmodellen unterstützt. Die Technologie stellt sicher, daß jede Veränderung eines Datums innerhalb der applikationsübergreifenden Datenbasis markiert wird. Damit können die Applikationen diese Technologie benutzen, um Veränderungen zu erkennen. Hat eine Applikation die Veränderung bemerkt, dann übernimmt diese den neuen Wert und startet anschließend die Neudarstellung des Fachmodells mit dem manipulierten Wert automatisch.

Der Anwender hat bei dieser Arbeitsweise lediglich eine passive Rolle. Er hat keine Möglichkeit, den Integrationsprozeß zu steuern, sondern bekommt immer bedingt korrekte Resultate vorgelegt. Die bedingt korrekten Resultate ergeben sich aus der Unvollständigkeit der Modelle nach dem Import der Daten kombiniert mit der Gegebenheit, daß die Aktionen nicht jede Möglichkeit der Instanzenerzeugung entsprechend der individuellen Vorstellungen der Anwender unterstützen können. Die Anwender bleiben aber für die Richtigkeit der Resultate verantwortlich, wobei die Überprüfung des Resultats durch das Vorgehen, die Erzeugung des komplexen Modells in einem Schritt, erschwert wird.

Ein weiterer Nachteil ergibt sich aus dem Umstand, daß diese Arbeitsweise Festlegungen benötigt, welche die Struktur und die Semantik der austauschbaren Daten umfassen. Dadurch können nur Daten ausgetauscht werden, die Gegenstand der Festlegungen sind. Der Anwender muß weiterhin die zugestellten Planungsunterlagen analysieren und die Informationen herausfiltern, die vom verwendeten Algorithmus nicht beachtet wurden, aber in seiner Planungsunterlage berücksichtigt werden müssen.

Im gewissen Umfang erlaubt der Einsatz von Integrationssoftware eine Verbesserung der Planungsqualität, trotz der Einschränkungen in Bezug auf die Flexibilität, d.h. die verwendbaren Möglichkeiten der Modellierung und des primitiven Ansatzes bei der Überprüfung der Korrektheit. Der effiziente Einsatz der Integrationssoftware ist aber im wesentlichen von den gegebenen Einsatzkriterien abhängig. Je mehr das Fachmodell, das mittels der Algorithmen der Integrationssoftware erzeugt wurde, mit den gewünschten Vorstellungen des Anwenders übereinstimmt, um so effizienter kann der Anwender mit der Integrationssoftware arbeiten. Für die effiziente Implementierung der Algorithmen ergibt sich die Bedingung, daß die Struktur wie auch Topologie der modellierten Bauelemente in allen Planungsunterlagen weitestgehend übereinstimmt.

Zwischen diesen beiden Arbeitsweisen gibt es noch eine Reihe weiterer, die zwar das Übernehmen von Daten erlauben aber keine Unterstützung für die Wahrung der Konsistenz anbieten. Beispielsweise ist hier für CAD-Applikationen der Datenaustausch via DXF zu nennen. Da diese Varianten den Einsatz von Integrationssoftware kombinieren, werden demnach auch nur die Arbeitsweisen der Anwender kombinert. Die Varianten erzeugen keine neue Arbeitsweisen für den Anwender.

3 Konzeptionelle Basis des Integrationsgedankens

Im Abschnitt 2.1 auf Seite 23 wurde folgende Ausgangssituation beschrieben. Jede Information im Rechner wird als Datum in einem Bereich des Speichers abgelegt. Dabei wird die Information in einem binären Code abgebildet. Die Anzahl Bits sowie die Art der binären Kodierung wird durch einen Typnamen festgelegt. Informationen sind somit typisierte Werte in Binärform ohne jegliche Semantik. Instanziiert belegt das Datum eine Position im Speicher.

Die Daten werden strukturiert abgelegt, wobei der Entwickler die Datenstrukturen sowie die Instruktionsketten festlegt, die auf den Datenstrukturen operieren. Der Anwender benutzt die bereitgestellten Instruktionsketten für das Erstellen seiner Planungsunterlage und kann somit durch das Aktivieren einer gewünschten Instruktionskette bestimmen, welche Daten zu erzeugen sind.

Für die Aufgaben der Integration spielen die modellierten Daten in den Anwendungen die dominante Rolle. Auf die implementierten Methoden der Anwendungen kann verzichtet werden. Zu den Aufgaben der Integration gehört es nicht, die spezifischen Methoden einer Applikation zu benutzen, um die Objekte der gerade bearbeiteten Planungsunterlage auf den modellierten Ausschnitt der realen Welt anzupassen.

Gegenstand des Integrationsprozesses ist es, eine Teilmenge von Informationen aus bestehenden Planungsunterlagen für eine darauf aufbauende Leistung zur Verfügung zu stellen sowie die dabei auftretenden Abhängigkeiten nachvollziehbar abzubilden. Im Abschnitt 2.2 wurden die aktuellen Arbeitsweisen aus dem Blickpunkt des Anwenders klassifiziert und ihre Vor- und Nachteile dargestellt. Die Arbeitsweise, die dieser Arbeit zugrunde liegt, verbindet die Vorteile der interaktiven Arbeitsweise, die den flexiblen Umgang der Daten sowie das iterative Vorgehen unterstützt, mit den Vorteilen der passiven Arbeitsweise, die den Einsatz von Integrationssoftware erlaubt.

3.1 Interaktive Arbeitsweise mit Integrationssoftware

Die interaktive Arbeitsweise basiert auf der aktiven Mitarbeit des Anwenders im Integrationsprozeß sowie auf einem iterativen Vorgehen des Anwenders bei der Bearbeitung der Integrationsaufgaben. Den Anwendern wird eine Menge von allgemeingültigen Aktionen zur Verfügung gestellt, die der Anwender für die Realisierung einer Integrationsaufgabe einsetzen kann.

Die Übernahme von Informationen muß demnach auf die Übernahme von Daten im Rechner aufbauen, wobei der Anwender die entsprechende Information bestimmt. Dazu kann er beispielsweise ein bestimmtes Datum auswählen, oder eine beliebige Menge von ausgewählten Daten benutzen, um das gewünschte Datum aus dieser Menge herzuleiten bzw. zu berechnen. Die Möglichkeiten zur Bestimmung der gesuchten Information muß die gleiche Mächtigkeit bieten wie die Möglichkeiten, die bei der traditionellen Arbeitsweise zum Einsatz kommen. Die passive Arbeitsweise mit Integrationssoftware verlangt eine Übernahme von Daten ohne Beteiligung des Anwenders für automatisierbare und sich wiederholende Arbeitsvorgänge.

Die Arbeitsweise bedingt also eine Methodik, mit der der Anwender die Abstraktion der Daten auf die gewünschte Ebene durchführen kann. Abfragesprachen erlauben auf der Basis einer

Grammatik Manipulationen an relational orientierten Datensätzen durchzuführen, ohne einen Compiler (Linker) vom Aufstellen der Abfrage bis zum Abarbeiten der Abfrage einzubeziehen. Die Mächtigkeit solch einer Grammatik ist von den möglichen Operationen abhängig, die die verwendeten Datentypen bereitstellen.

Da die möglichen atomaren Datentypen bekannt sind sowie auch die Regeln, mittels derer bekannte Datentypen zu neuen kombiniert werden können, kann man eine Sprache angeben, deren Operatoren Daten von dynamisch erzeugten Datentypen verarbeiten können und die geforderte Mächtigkeit dem Anwender bereitstellen. Die beiden noch offenen Fragen sind: Wie werden die Daten mit ihrer Typbeschreibung und ihrem Zustand ermittelt, und wie erfolgt die Präsentation der Daten, damit der Anwender seine Abfragen formulieren kann?

Die Auswahl der verwendeten Daten erfolgt durch den Anwender, indem er die applikationsspezifischen Identifizierungsmöglichkeiten benutzt. Neue Entwicklungen von Kommunikationskonzepten stellen Protokolle bereit, um ein Datum selbst zu fragen, welche Typbeschreibung es besitzt. Ist die Typbeschreibung bekannt, d.h. alle Attributnamen mit ihren Datentypen, können die Werte der Attribute erfragt werden.

Für das Darstellen der Daten muß beachtet werden, daß die fachspezifische Darstellung der Daten nur in der jeweiligen Applikation möglich ist. Die Abfragesprache muß demnach auf eine eigene Darstellungskomponente aufbauen. Datenbanken, die auch umfangreiche Datenbestände in einem sogenannten Browser darstellen müssen, benutzen hierzu die hierarchische oder tabellarische Anordnung von Daten.

Bei der hierarchischen Darstellung werden zusammengesetzte Daten als Baum dargestellt. Tupel- und Listentypen fügen neue Knoten ein, atomare Datentypen bilden die Blätter des Baumes (siehe Abbildung 3.1).

Bei der tabellarischen Darstellung wird der entsprechende Datentyp als Tabellenkopf, die einzelnen Objekte als Zeilen der Tabelle ausgegeben. Tupeltypen erzeugen neue Spalten in einer übergeordneten Spalte, Listentypen erzeugen mehrere Zeileneinträge innerhalb einer Objektzeile. Atomare Datentypen belegen eine Zeile in einer Spalte. Dieses Prinzip wird auch in objektorientierten Datenbanken verwendet und ist als geschachtelte Relation bekannt (siehe Abbildung 3.2).

Das empfohlene Vorgehen beginnt mit der Auswahl der Daten in den entsprechenden Präsentationen der bereitgestellten Planungsunterlagen. Die ausgewählten Objekte werden den Anwendern beispielsweise tabellarisch dargestellt. Der Anwender erstellt individuelle Abfragen, mit dem Ziel, die übernommenen Daten in die gewünschte, vom Zielsystem vorgeschriebene Struktur abzubilden. Ist die gewünschte Struktur erreicht, können die Daten in das Zielsystem eingefügt werden.

Dieses Vorgehen entspricht der iterativen Arbeitsweise, bei welcher nachvollziehbare Veränderungen an der gerade bearbeiteten Planungsunterlage auftreten. Des weiteren erlaubt dieses Vorgehen einen flexiblen Umgang mit den Informationen, da die Übertragung von Daten an keinen Algorithmus gebunden ist, sondern vom Anwender in Form von Abfragen vorgegeben wird. Der Anwender kann durch die Auswahl bestimmmter Objekte und durch individuelle Abfragen den Inhalt des Integrationsprozesses beeinflussen. Die Auswahl der Objekte kann beliebig oft wiederholt werden.

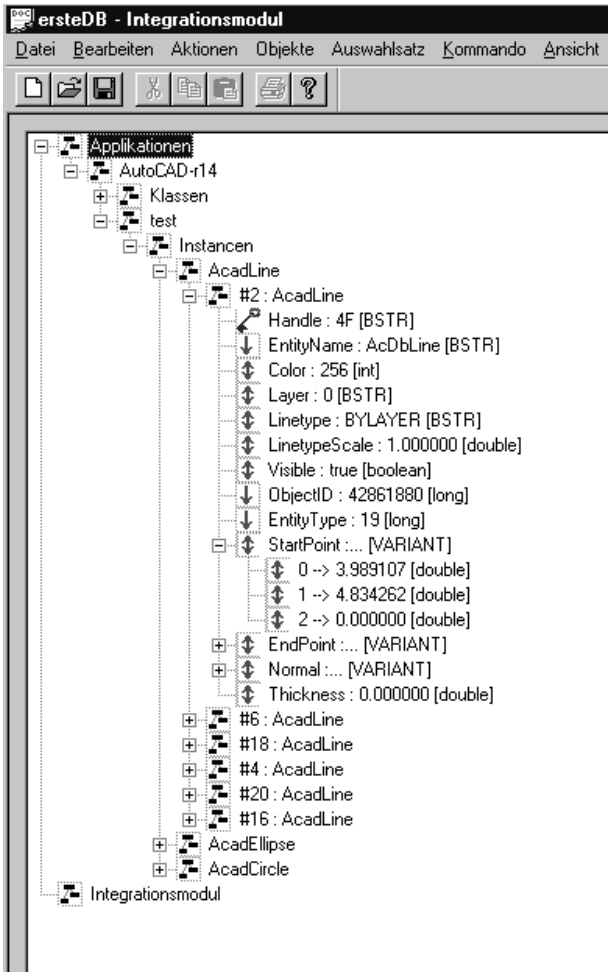


Abbildung 3.1: Hierarchische Darstellung der Instanzen

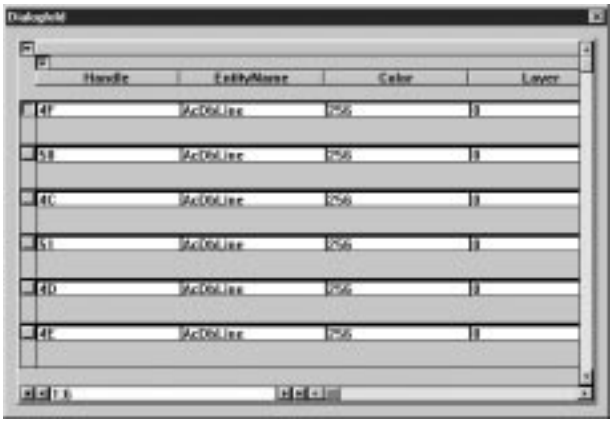


Abbildung 3.2: Tabellarische Darstellung der Instanzen

Automatisiert werden kann der Prozeß, indem ausgewählte Objekte aus den Planungsunterlagen übernommen und mittels persistent vorhandener Abfragen in die entsprechende Struktur des Zielsystem abgebildet werden. Die einzelnen Abfragen können in einem Makro gesammelt und so nacheinander abgearbeitet werden. Am Ende muß der Anwender nur noch das Zielmodell angeben, in das die einzelnen überarbeiteten Objekte einzufügen sind.

Die rechnerinterne Übernahme von Daten bedingt, daß die Präsentation der bereitgestellten Planungsunterlage in digitaler Form vorliegt. Da die Präsentation nur in der entsprechenden Applikation dargestellt werden kann, muß demnach die Applikation auch für die Anwender verfügbar sein.

Ungewöhnlich für den Anwender ist die Darstellung der Daten in einem Baum oder in einer Tabelle. Anhand dieser Darstellung muß er entscheiden, welche Attribute er von den selektierten Objekten benötigt und wie diese weiter zu verwenden sind. Die semantische Interpretation der Strukturbeschreibungen ist Aufgabe des Anwenders. Diesen Nachteil muß der Anwender in Kauf nehmen, wenn er in den Integrationsprozeß aktiv eingebunden werden will. Dabei können Hilfe-Texte oder aussagekräftige Bezeichnungen der Attribute als Hilfestellungen anerkannt werden.

Die semantische Interpretation wird leichter, wenn die Daten der Denkweise des Anwenders entsprechen. Objektorientierte Modelle, die ihre Objekte parametrisch in einfachen Strukturen beschreiben, bilden die ideale Ausgangsposition für diese Arbeitsweise. Diese Objekte lassen sich mit wenigen und einfachen Abfragen in andere Modelle überführen.

Für die zweite Integrationsaufgabe benötigt der Anwender Informationen, um eine Aktualisierung seiner Planungsunterlage effektiv durchführen zu können. Aufgabe der Software kann es nur sein, den Anwender über aufgetretene Veränderungen in anderen Planungsunterlagen zu informieren.

Das Bereitstellen dieser Informationen bedingt eine Technologie, die im Fall der Aktualisierung einer Planungsunterlage, die aufgetretenen Änderungen erfaßt und auf Basis dieser Veränderungen in der aktualisierten Planungsunterlage die Planungsunterlagen ermittelt, die auf Grund dieser Veränderungen wiederholt aktualisiert werden müssen. Die Auswahl der entsprechend notwendigen Aktionen in den zu aktualisierenden Planungsunterlagen muß durch den Anwender (Eigentümer der Planungsunterlage) erfolgen.

Das Erkennen der Veränderungen und die persistente Modellierung der Abhängigkeiten zwischen den einzelnen Planungsunterlagen bedingt, daß die Granularität der kleinsten betrachtbaren Einheit festzulegen sind.

Zweckmäßig ist es, die Granularität entsprechend der menschlichen Denkweise festzusetzen. Demnach entspricht die kleinste betrachtbare Einheit für die Wahrung der Konsistenz den Objekten einer Planungsunterlage. Diese Festlegung benötigt keine weitere Mapping-Mechanismen, da die Dokumente der Planungsunterlagen nach gegenwärtigem Stand der Technik objektorientiert erstellt werden. Des weiteren kann auf die Identifizierung der Objekte in den Dokumenten zurückgegriffen werden für folgende Aufgaben:

- Bestimmung von manipulierten Objekten,
- persistente Speicherung von Abhängigkeiten zwischen Objekten verschiedener Planungsunterlagen durch eine auf den Objektidentifikatoren basierende Relation.

Die aktive Arbeitsweise des Anwenders, der für die Realisierung der Integrationsaufgaben Softwarewerkzeuge einsetzt, bedingt, daß seine Planungsunterlage anderen Anwendern in Form von digitalen Dokumenten zur Verfügung gestellt werden. Aus organisatorischen Gründen ist es zweckmäßig, die einzelnen Dokumente in einen Container einzubetten.

3.2 Notwendige Aktionen im Integrationsprozeß

Die Spezifikation der Aktionen ergibt sich aus den Aufgaben der Integration sowie durch die Arbeitsweise, die in dieser Arbeit untersucht wird. Die Aktionen lassen sich in verschiedene Gruppen einteilen.

- Aktionen zur Initialisierung von Modell und Applikationen in einem Container
- Aktionen für die Übernahme von Daten
- Aktionen für die Wahrung der Konsistenz

3.2.1 Aktionen zur Initialisierung von Modell und Applikationen

Während des Integrationsprozesses treten Vorgänge auf, die sich ausschließlich auf die Bearbeitung von Planungsunterlagen beziehen. Zum einen werden neue Planungsunterlagen auf der Basis von übernommenen Informationen erzeugt. Zum anderen erzeugt der Planer Planungsunterlagen, die zum Zeitpunkt der Erzeugung erst einmal unabhängig von den bereits vorhandenen Planungsunterlagen sein können. Die parallele Arbeitsweise der Planer erfordert jedoch, daß alle Planungsunterlagen in einem Container zusammengetragen werden, wo sie voneinander unabhängig bearbeitet werden können.

Die voneinander unabhängige Existenz der Planungsunterlagen führt zu redundanten Informationen, d.h. jede Planungsunterlage muß über einen persistenten unabhängigen Block verfügen, der die entsprechenden Daten der Planungsunterlage beinhaltet. Die Aktionen für die Erzeugung und für das Öffnen von Planungsunterlagen muß demnach auf der Ebene 'erzeuge einen neuen persistenten Block' oder 'öffne einen persistenten Block' im Container liegen. Diese Blöcke werden üblicherweise als *embedded objects* des Containers bezeichnet.

Für die Bearbeitung der Integrationsaufgaben müssen die Daten der einzelnen Planungsunterlagen dem Anwender zur Verfügung gestellt werden. Die notwendige Präsentation der Daten ergibt sich aus den geforderten Richtlinien des hierfür verwendeten Modellierungskonzeptes. Da nur die Daten der Planungsunterlagen in dem Container gespeichert werden und nur die entsprechende Applikation über das implementierte Wissen verfügt, wie aus den Daten die Präsentation zu berechnen ist, muß der Container die entsprechende Applikation kennen, um gegebenenfalls einen Zugriff auf die Applikation generieren zu können.

Demnach besteht zwischen den Applikationen und den Planungsunterlagen eine Beziehung. Jede Planungsunterlage wird von genau einer Applikation bearbeitet, jede Applikation kann beliebig viele Planungsunterlagen assoziieren. Die einzelnen Applikationen mit ihren Planungsunterlagen lassen sich somit hierarchisch strukturieren.

Daraus resultieren folgende Aktionen:

- Hinzufügen einer Applikation (deren Referenz) zu dem Container.
- Erzeugen einer neuen Planungsunterlage (embedded object) durch die Angabe der Applikation und Hinzufügen der erzeugten Planungsunterlage zu dem Container.
- Öffnen einer bestehenden Planungsunterlage (embedded object) durch die Angabe der Applikation und seiner Identität (Dateiname) und Hinzufügen der geöffneten Planungsunterlage zum Container.
- Entfernen einer Planungsunterlage.
- Entfernen einer Applikation (deren Referenz) aus dem Container.

Da jede Planungsunterlage und Applikation im Container eindeutig vom Anwender ausgewählt werden muß, sind Aktionen für die Zuordnung einer Identifizierung sowie für das Finden einer Planungsunterlage und Applikationen mittels der Identifikation notwendig.

Nachdem die Aktionen angegeben wurden, die für das Einfügen und das Löschen einer Planungsunterlage verantwortlich sind, sind jetzt die Aktionen für die Manipulation einer Planungsunterlage von Interesse. Die Manipulation einer Planungsunterlage kann nur im geöffneten Modus erfolgen.

Für die Bearbeitung einer Planungsunterlage sollte der Container dem Anwender zwei Möglichkeiten anbieten. Für kleinere Veränderungen ist es zweckmäßig, die Überarbeitung direkt im Container durchzuführen. Für umfangreiche Überarbeitungen ist es hilfreich, den Datenbestand der betrachteten Planungsunterlage aus dem Container zu exportieren und später die überarbeitete Planungsunterlage wieder in den Container einzufügen.

Demnach werden noch folgende Aktionen benötigt:

- Öffnen/Speichern einer Planungsunterlage.
- Export/Import einer Planungsunterlage.

3.2.2 Aktionen für die Übernahme von Daten

Die interaktive Arbeitsweise des Anwenders führt zu einem iterativen Integrationsprozeß, der durch den Anwender mittels allgemeingültiger Aktionen gesteuert wird. Die objektorientierte Denkweise des Anwenders ist ausschlaggebend für die Spezifikation der Aktionen.

Für den Anwender spielen die betrachtbaren Objekte in den einzelnen Planungsunterlagen die dominante Rolle. Aus diesem Grund müssen nur die Objekte, die eine grafische[1] Präsentation in der Planungsunterlage besitzen, vom Integrationsansatz betrachtet werden. Die grafische Präsentation hängt von der Belegung der einzelnen Attributwerte des Objektes sowie von den aktuellen

[1] Grafisch bedeutet in diesem Zusammenhang darstellbar in der Benutzeroberfläche

Beziehungen zu anderen Objekte ab. In diesem Zusammenhang sind beispielsweise Bauteile wie Wand, Fenster, Dach und Treppe sowie Positionen in der Statik oder im Leistungsverzeichnis die betrachtbaren Objekte.

Gegenstand dieses Prozesses ist es somit, eine Reihe von Objekten einer Planungsunterlage durch den Anwender auswählen zu lassen, die die gleiche Abbildung bei der Bearbeitung der Integrationsaufgabe durchlaufen. Die selektierten Objekte können verwendet werden, um bestimmte Objektattribute bei der Erzeugung von Instanzen in einer anderen Planungsunterlage zu besetzen. Da man nicht davon ausgehen kann, daß die Attribute der Objekte von Hause aus den Ansprüchen der Zielapplikation genügen, muß der Integrationsansatz die Manipulation der Attribute auf Objektebene unterstützen.

Für das Auswählen der Objekte spielen die Möglichkeiten für die Identifizierung der Objekte in einer Planungsunterlage eine relevante Rolle. Die implementierten Möglichkeiten der Objektidentifizierung in den einzelnen Applikationen, die den Richtlinien des Modellierungkonzeptes entsprechen, sind hierfür ausschlaggebend. Demnach stehen dem Anwender optimale Verfahren zur Verfügung.

Nach der Auswahl der Objekte erfolgt deren Manipulation. Gegenstand dieses Arbeitsschrittes ist es, die Inhalte der ursprünglichen Objekte auf die gewünschten Inhalte, die von der Zielapplikation benötigt werden, abzubilden. Hierzu sind Operationen notwendig, die entweder die Typbeschreibung, den Zustand oder beides der ursprünglichen Objekte verändern. Die Mächtigkeit, die diese Operationen dem Anwender für die Manipulation der Objekte bieten, muß mit den traditionellen Möglichkeiten einer Bearbeitung übereinstimmen. Zweckmäßig ist es, eine Reihe von allgemeingültigen Operationen anzugeben. Jede Operation führt eine überschaubare Veränderung an einer Menge gleichwertiger Objekte durch. Durch die geschickte Kombination der Operationen lassen sich auch umfangreiche Manipulationen durchführen.

Das Kombinieren der einzelnen Operationen ist nur möglich, wenn das Resultat einer Operation für die Eingabe in andere Operationen benutzt werden kann. Demnach müssen alle Operationen Element eines Akteurs sein, der auch das Resultat einer Operation darstellt. Das Zusammenfassen von Objekten zu einem Auswahlsatz erfüllt diese Bedingung, wenn die einzelnen Auswahlsätze die Operationen beinhalten, und die Operationen ihre manipulierten Objekte wieder zu einem Ergebnisauswahlsatz zusammenfassen.

Die Übernahme von Daten beginnt mit dem Auswählen einer bestimmten Planungsunterlage. Durch die Zuordnung, welche Applikation diese Planungsunterlage bearbeitet, wird die Planungsunterlage geöffnet und präsentiert sich dem Anwender in der fachspezifischen Notation. Die nächste Aktion beinhaltet das Auswählen der gesuchten grafischen Objekte, die für die betrachtete Integrationsaufgabe benötigt werden. Für die ausgewählten Objekte wird der Datentyp sowie der Zustand ermittelt und separat nachgebaut, wobei für jeden unterschiedlichen Datentyp der ausgewählten Objektmenge ein neuer Auswahlsatz erstellt wird. Das Ergebnis des ersten Arbeitsschrittes bilden die generierten Auswahlsätze, deren Objekte tabellarisch oder hierarchisch dargestellt werden.

Mittels dieser grafischen Präsentationen kann der Anwender die notwendigen Operationen zusammenstellen, die die gewünschte Objektbeschreibung oder den Objektzustand aus den selektierten Objekten abstrahieren und das Resultat in einem Auswahlsatz mit manipulierten Objekten den Anwendern zur Verfügung stellen. Die dafür benötigte Menge aller Operationen ist dabei nicht von einer speziellen Übernahme von Daten zweier Planungsunterlagen abhängig. Der Auswahl-

satz mit den manipulierten Objekten bildet die Grundlage für die Erzeugung der Instanzen in einer anderen Planungsunterlagen.

Einschub Definitionen Strukturbeschreibung:

Jedes grafische Objekt in einem Computermodell basiert entsprechend dem objektorientierten Paradigma auf einer Strukturbeschreibung, einem Zustand und einer Identifikation. Die Strukturbeschreibung wird über einen in der Applikation eindeutigen Bezeichner angegeben. Die Regeln für die Erstellung eines neuen, noch unbekannten Datentyps in den Applikationen erfolgen nach folgenden formalen Axiomen[2].

- Definition: semantische Umschreibung
  U sei eine Menge von Umschreibungen von modellierten Eigenschaften. Ein Element $u \in U$ heißt Bezeichner.

- Definition: Datentyp:

  1. Die Menge $S^0 = \{void, Integer, Long, Double, Float, String, Bool\}$ enthält gültige atomare Datentypen.
  2. Die Menge $S^{i+1} = S^i \cup \{s^{i+1}\}$ kann aus der Menge S^i durch das Hinzufügen eines neuen Datentyps s^{i+1} gebildet werden, wenn für s^{i+1} folgendes gilt:
     - $s^{i+1} := \mathcal{K}$, wenn $\mathcal{K} \in \{\mathcal{K}_T, \mathcal{K}_L, \mathcal{K}_M, \mathcal{K}_{Ref}\}$ ist. Dabei stellen $\mathcal{K}_T, \mathcal{K}_L, \mathcal{K}_M$ die Möglichkeiten dar, neue zusammengesetzte Datentypen zu konstruieren.
     - $s^{i+1} \notin S^i$.

  Die möglichen Konstruktionen setzen sich wie folgt zusammen:

  - Tupel $\mathcal{K}_T$:
    sind $s_0, \ldots, s_l$ gültige Datentypen in S^i, $u_0, \ldots, u_l$ gültige Umschreibungen in U, und $< u_i, s_i, b_i >$ gültige Attribute, dann wird mit
    $\mathcal{K}_T$:=Tupel Of($\{< u_0, s_0, b_0 >, < u_1, s_1, b_1 >, \ldots, < u_l, s_l, b_l >\}$) ein gültiger Datentyp erzeugt.
    Dabei müssen die Bezeichner zueinander eindeutig sein, d.h.:
    $\forall\, i \in [0, \ldots, l]$ und $\forall\, k \in [0, \ldots, l]$ gilt $i \neq k \Longleftrightarrow u_i \neq u_k$ mit $l \in \mathbb{N}$.

  - Liste $\mathcal{K}_L$:
    ist s ein gültiger Datentyp in S^i, dann erzeugt
    $\mathcal{K}_L$:=List Of($\{s\}$) einen gültigen Datentyp.

  - Menge $\mathcal{K}_M$:
    ist s ein gültiger Datentyp in S^i, dann erzeugt
    $\mathcal{K}_M$:=Set Of($\{s\}$) einen gültigen Datentyp.

  - Beziehungen $\mathcal{K}_{Ref}$
    ist s ein gültiger Datentyp in S^i, dann erzeugt
    $\mathcal{K}_{Ref}$:=Relation To($\{s\}$) einen gültigen Datentyp.

[2] vergleiche hierzu auch Abschnitt 2.1.2 auf Seite 27

Die Menge $S = \bigcup_i S^i \backslash \{void\}$ beinhaltet alle möglichen Datentypen, die für die Erzeugung zusammengesetzter Datentypen verwendet werden können.

- Definition Wertebereich:
  Für jeden atomaren Datentyp gibt es eine Menge, die den Wertebereich des Datentyps festlegt. Die möglichen Elemente wurden definiert.
  Die Zuordnung des Wertebereichs bei einem Tupeltyp ergibt sich aus dem Kreuzprodukt der Wertebereiche der einzelnen Komponenten, bei einer Liste aus der Kleenschen Hülle und bei einer Menge aus der Potenzmenge des zugrundeliegenden Wertebereichs.

- Definition Attribut, Wertebereich und Attributwert:
  Das geordnete Tripel $a :=< u, s, b >$ mit $u \in U \wedge s \in S \wedge b \in \{\text{key,read-only,normal}\}$ wird als Attribut bezeichnet. Die Schalter beinhalten folgende Semantik:

key	Das Attribut wird für die Identifizierung des Objektes in der Applikation verwendet.
read-only	Das Attribut kann nicht verändert werden.
normal	Das Attribut besitzt keine Einschränkungen in Bezug auf den Zugriff.

  Ein Attribut repräsentiert eine Eigenschaft des zugehörigen Modells. Die Menge aller Attribute wird als Attributmenge A bezeichnet. Die Funktion $dom : S \rightarrow W^*$ ordnet jedem Datentyp $s \in S$ einen Wertebereich zu, wobei W^* das Mengensystem alle verfügbaren Wertebereiche darstellt. Ein Element des Wertebereiches $w \in W$ nennt man Attributwert.

- Definition Objekt, Auswahlsatz und Datenbank
  Ein Objekt des Integrationsansatzes $o :\Leftrightarrow < \sigma, s, z(s) >$ ist ein geordnetes Tripel, bestehend aus der Objektidentität (σ), dem Datentyp (z) und einer Abbildung $z : S \rightarrow W^*$, wobei $z(s) \in dom(s)$ gilt. Da jedes Objekt im Integrationsansatz über eine entsprechende grafische Präsentation verfügt, die sich im Normalfall aus einer Menge von mindestens einem Attribut herleiten läßt und des weiteren mindestens ein Attribut für die Identifizierung des Objektes in der Planungsunterlage (siehe Abschnitt 3.5 auf Seite 67), kann davon ausgegangen werden, daß die zu erwartenden Objekttypen Tupelkonstruktionen entsprechen. Demnach kann für die Definition von Objekten folgende zwei Schreibweisen $< \sigma, s, z(s) > \equiv < \delta, \mathcal{K}_T(\{A\}), z(\mathcal{K}_T(\{A\})) >$ angegeben werden. Eine Menge von typgleichen Objekten wird als Auswahlsatz $\wp = \{o_0, o_1, \ldots, o_n\}$ bezeichnet. Eine Menge von Auswahlsätzen definiert eine Datenbank.

Das Manipulieren der Objekte in die vorgeschriebene Form der Zielanwendung kann eine Veränderung der Objektbeschreibung oder des Zustandes des Objektes nach sich ziehen. Hierzu stehen generische und typspezifische Operationen zur Verfügung, die auf den obigen abstrakten Definitionen aufbauen.

Generische Operationen sammeln bestimmte Objekte in einer Ergebnismenge, die durch entsprechende Bedingung aus einer Auswahlmenge extrahiert wurden. Sie sind in der Lage, neue Objektbeschreibungen zu erzeugen und neue Objekte zu instanziieren.

Typspezifische Operationen sind von zugrundeliegenden Datentypen abhängig und regeln die Handhabung der darauf aufbauenden Daten.

	Integer / Long	**Double / Float**	**String**	**Bool**
Integer Long		keine Einschränkung	Zifferndarstellung	$> 0 \rightarrow$ true sonst $\rightarrow$ false
Double Float	math. Runden		Zifferndarstellung	$> 0 \rightarrow$ true sonst $\rightarrow$ false
String	Zifferndarstellung	Zifferndarstellung		"true"$\rightarrow$ true "false"$\rightarrow$ false sonst $\rightarrow$ error
Bool	true $\rightarrow$ 1 false $\rightarrow$ 0	true $\rightarrow$ 1.0 false $\rightarrow$ 0.0	true $\rightarrow$ "true" false $\rightarrow$ "false"	

Tabelle 3.1: Konvertierungsregeln atomarer Datentypen

Typspezifische Operationen

Die typspezifischen Operationen werden für die Formulierung von Bedingungen, für die Berechnung von Werten oder für die Zuweisung des Wertes an ein bestimmtes Attribut benötigt. Die einzelnen Operationen lassen sich in zwei Kategorien einteilen, je nachdem ob sie auf atomaren oder zusammengesetzten Datentypen operieren. Alle binäre Operationen verlangen, daß die Datentypen der zu verknüpfenden Werte typkompatibel sind.

- **atomare Datentypen**

Vergleichsoperationen auf atomaren Datentypen:
Zu den Vergleichsoperationen (siehe Tabelle 3.2) zählen:$\{<, \leq, =, \geq, \neq\}$
Die Operatorsymbole gelten für die atomaren Datentypen im allgemein mathematischen Sinn.

Zuweisungsoperationen auf atomaren Datentypen:
Zu den Zuweisungsoperationen (siehe Tabelle 3.4) zählen:$\{=, + =, - =, / =, * =\}$
Die Operatorsymbole gelten im ursprünglich programmiertechnischen Sinn.

Konvertierungsoperationen auf atomare Datentypen:
Die Konvertierungsregeln werden automatisch (siehe Tabelle 3.1) angewendet, falls Attributwerte mit unterschiedlichen Datentypen als Parameter einer Operation aufgerufen werden.
Die Konvertierung eines String-Wertes in einen anderen atomaren Datentypwert basiert auf den üblichen Regeln in den Programmiersprachen.

Spezielle Operationen atomarer Datentypen:
Die speziellen Operationen auf numerischen Datentypen erlauben die Addition, Subtraktion sowie die Multiplikation zweier Zahlen. Eine Ausnahme bildet die Division, die nur bei den gebrochenen Zahlen den Quotienten im üblichen mathematischen Sinn berechnet. Handelt es sich beim Dividenden wie auch beim Divisor um eine ganze Zahl, dann berechnet der Divisionsoperator den ganzzahligen Anteil. Zur speziellen Operation von Stringwerten zählt nur die Addition, die den zweiten Summand an das Ende des ersten Summanden anhängt. Zu den speziellen Operationen von Boolwerten gehören die logischen Operationen AND, OR und NOT (siehe Tabelle 3.3).

Zusätzlich unterstützt jeder atomare Datentyp noch eine spezielle Operation, den **Initialisierer**. Wann immer eine neue Instanz des Datentyps erzeugt wird, bekommt die Instanz einen voreingestellten Wert zugewiesen. Bei Integer ist das 0, bei Double 0.0, bei Zeichenketten ist das die leere Zeichenkette und bei Bool ist es der Wert FALSE. Die Einstellungen können nicht verändert werden.

- **zusammengesetzte Datentypen**

Die Vergleichsoperatoren für die zusammengesetzten Datentypen lassen sich nur eingeschränkt auf die atomaren Operationen auf den einzelnen Komponenten zurückführen. Im allgemeinen sind nur der Test auf Gleichheit und die entsprechende Verneinung definiert. Somit sind zwei Tupelwerte gleich, wenn die einzelnen Komponenten der beiden Tupelwerte zueinander gleich sind. Bei Mengenwerten muß gelten, daß der eine Mengenwert Teilmenge der anderen ist und umgedreht. Für Listen wird die Gleichheit über die einzelnen Elemente definiert. Sind die beiden ersten Elemente beider Listen gleich, dann wird das zweite Element untersucht. Sind diese beiden gleich, wird mit den nächsten Listenelementen fortgesetzt. Demnach sind zwei Listen gleich, wenn diese beiden Listen die gleiche Anzahl von Elementen beinhalten und die einzelnen Elemente selbst zueinander gleich sind.

Für Mengen ist noch die Vergleichsrelation 'kleiner als' definiert. Die Relation wird über die echte Teilmengenbeziehung zwischen zwei Mengen definiert. Aus diesen beiden Relationen lassen sich für Mengen alle weiteren Vergleichsrelationen ableiten. Für die Liste und den Tupel sind diese Relationen nicht definiert.

Für Mengen sind die arithmetischen Operationen +, - und / definiert. Mit den einzelnen Operationen wird die Vereinigung, die Differenz und der Durchschnitt angegeben.

Für Listen existieren noch die Operationen + und -. Bei Listen beinhalten die Operationen das Einfügen eines neuen Listenelementes am Ende der Liste oder das Löschen des angegebenen Listenelementes.

Jeder zusammengesetzte Datentyp unterstützt die einfache Zuweisung =. Je nachdem welche arithmetischen Operationen zugelassen sind, sind auch die entsprechenden Zuweisungsoperationen gültig.

Konvertieren läßt sich ohne Einschränkung eine Menge in eine Liste. Wird eine Liste in eine Menge konvertiert, dann werden entsprechend der Mengendefinition alle mehrfach enthaltenen Datensätze einer Liste entfernt.

Die letzte allgemeine Operation kommt zum Einsatz, sobald eine neue Instanz von einem zusammengesetzten Datentyp erzeugt wird. Der Initialisierer von den zusammengesetzten Datentypen ist verantwortlich, daß die Initialisierer der zugrundeliegenden Komponentendatentypen angewendet werden.

Zusätzlich erfordert die Spezifikation, daß eine Vergleichsoperation für die Typgleichheit $\stackrel{T}{=}$ zur Verfügung steht. Diese Operation wird für die Konsistenzprüfung benötigt. Zwei Typen sind gleich, wenn sie beide vom gleichen atomaren Datentyp sind, oder wenn ihre Komponentenmengen, aus denen sie zusammengesetzt sind, zueinander gleich sind.

Für jeden zusammengesetzten Datentyp gibt es noch spezielle Operationen. Die folgende Aufzählung listet die einzelnen Operationen in Abhängigkeit der Typkonstruktion auf:

- TUPEL:
  - Komponentenzugriff (Punktoperator) $\mathcal{K}_T(A).u = \begin{cases} s & < u,s,b > \in A \\ void & sonst \end{cases}$
  - Hinzufügen und Entfernen einer Komponente (eines Attributes) (ADD, REMOVE)
  - Durchschnitt, Vereinigung und Differenz zweier Tupeltypen ($\overset{\mathrm{T}}{\bigcup}, \overset{\mathrm{T}}{\bigcap}, \overset{\mathrm{T}}{\setminus}$) mit:
    - $\mathcal{K}_T(A_1) \overset{\mathrm{T}}{\bigcup} \mathcal{K}_T(A_2)$
      $= \mathcal{K}_T(\{< u,s,b > |\ s \in S \wedge (\mathcal{K}_T(A_1).u = s \wedge \mathcal{K}_T(A_2).u = void\} \cup$
      $\{< u,s,b > |\ s \in S \wedge (\mathcal{K}_T(A_2).u = s \wedge \mathcal{K}_T(A_1).u = void\} \cup$
      $\{< u,s,b > |\ \mathcal{K}_T(A_1).u = s_1 \wedge \mathcal{K}_T(A_2).u = s_2 \wedge (s = s_1 \overset{\mathrm{T}}{\bigcup} s_2) \in S\}$
      $)$
    - $\mathcal{K}_T(A_1) \overset{\mathrm{T}}{\bigcap} \mathcal{K}_T(A_2)$
      $= \mathcal{K}_T(\{< u,s,b > |\ s \in S^0 \wedge (< u,s,b > \in A_1 \wedge < u,s,b > \in A_2)\} \cup$
      $\{< u, s_1 \overset{\mathrm{T}}{\bigcap} s_2, b > | < u,s_1,b > \in A_1 \wedge < u,s_2,b > \in A_2 \wedge s_1, s_2 \notin S^0\}$
      $)$
    - $\mathcal{K}_T(A_1) \overset{\mathrm{T}}{\setminus} \mathcal{K}_T(A_2)$
      $= \mathcal{K}_T(\{< u,s,b > |\ s \in S^0 \wedge (< u,s,b > \in A_1 \wedge < u,s,b > \notin A_2)\} \cup$
      $\{< u, s_1 \overset{\mathrm{T}}{\setminus} s_2, b > | < u,s_1,b > \in A_1 \wedge < u,s_2,b > \in A_2 \wedge s_1, s_2 \notin S^0\}$
      $)$
  - Teilmenge zweier Tupeltypen

    $\mathcal{K}_T(A_1) \overset{\mathrm{T}}{\subset} \mathcal{K}_T(A_2) \leftrightarrow (\forall < u,s,b > \in A_1 \wedge s \in S^0 \wedge < u,s,b > \in A_2) \vee$
    $(\forall < u,s_1,b > \in A_1 \wedge s_1 \notin S^0 \wedge < u,s_2,b > \in A_2 \wedge s_1 \overset{\mathrm{T}}{\subset} s_2)$
  - Typgleichheit zweier Tupeltypen
    $\mathcal{K}_T(A_1) \overset{\mathrm{T}}{=} \mathcal{K}_T(A_2) \leftrightarrow \mathcal{K}_T(A_1) \overset{\mathrm{T}}{\subset} \mathcal{K}_T(A_2) \wedge \mathcal{K}_T(A_2) \overset{\mathrm{T}}{\subset} \mathcal{K}_T(A_1)$
  - Benennung eines Types $\mathcal{K}_T(A).toString$ = 'TUPEL OF'.

- LISTE:
  - Zugriff auf das erste, letzte, nächste oder vorhergehende Listenelement (Iteratoren) (FIRST, LAST, NEXT, PREVIOUS)
  - erstes Element der Liste (HEAD) und ohne das erste Element beinhaltende Restliste (TAIL)
  - Elementrelation (HASELEMENT)
  - Elementzugriff (Iteratoren) (COUNT, ITEM)
  - Durchschnitt, Vereinigung und Differenz zweier Listentypen ($\overset{\mathrm{T}}{\bigcup}, \overset{\mathrm{T}}{\bigcap}, \overset{\mathrm{T}}{\setminus}$) mit:
    - $\mathcal{K}_L(\{s_1\}) \overset{\mathrm{T}}{\bigcup} \mathcal{K}_L(\{s_2\}) = \mathcal{K}_L(\{s_1 \overset{\mathrm{T}}{\bigcup} s_2\})$
    - $\mathcal{K}_L(\{s_1\}) \overset{\mathrm{T}}{\bigcap} \mathcal{K}_L(\{s_2\}) = \mathcal{K}_L(\{s_1 \overset{\mathrm{T}}{\bigcap} s_2\})$

  - $\mathcal{K}_L(\{s_1\}) \overset{\mathrm{T}}{\setminus} \mathcal{K}_L(\{s_2\}) = \mathcal{K}_L(\{s_1 \overset{\mathrm{T}}{\setminus} s_2\})$
- Teilmenge zweier Listentypen
  $\mathcal{K}_L(\{s_1\}) \overset{\mathrm{T}}{\subset} \mathcal{K}_L(\{s_2\}) \leftrightarrow s_1 \overset{\mathrm{T}}{\subset} s_2)$
- Typgleichheit zweier Listentypen
  $\mathcal{K}_L(\{s_1\}) \overset{\mathrm{T}}{=} \mathcal{K}_L(\{s_1\}) \leftrightarrow s_1 \overset{\mathrm{T}}{=} s_2$
- Zugriff auf den Datentyp eines Listenelementes $\mathcal{K}_L(\{s\}).type = s$.
- Benennung eines Types $\mathcal{K}_L(A).toString =$ 'List Of'.
- Reduziere die Präsentation der referenzierten Objekte auf ihre Objektidentifikation (CreateReference) und konvertiere den Listentyp in einen Referenztyp um.

- Menge:
  - Elementrelation (hasElement)
  - Elementzugriff (Iteratoren) (Count, Item)
  - Durchschnitt, Vereinigung und Differenz zweier Mengentypen ($\overset{\mathrm{T}}{\bigcup}, \overset{\mathrm{T}}{\bigcap}, \overset{\mathrm{T}}{\setminus}$) mit:
    - $\mathcal{K}_M(\{s_1\}) \overset{\mathrm{T}}{\bigcup} \mathcal{K}_M(\{s_2\}) = \mathcal{K}_M(\{s_1 \overset{\mathrm{T}}{\bigcup} s_2\})$
    - $\mathcal{K}_M(\{s_1\}) \overset{\mathrm{T}}{\bigcap} \mathcal{K}_M(\{s_2\}) = \mathcal{K}_M(\{s_1 \overset{\mathrm{T}}{\bigcap} s_2\})$
    - $\mathcal{K}_M(\{s_1\}) \overset{\mathrm{T}}{\setminus} \mathcal{K}_M(\{s_2\}) = \mathcal{K}_M(\{s_1 \overset{\mathrm{T}}{\setminus} s_2\})$
  - Teilmenge zweier Mengentypen
    $\mathcal{K}_M(\{s_1\}) \overset{\mathrm{T}}{\subset} \mathcal{K}_M(\{s_2\}) \leftrightarrow s_1 \overset{\mathrm{T}}{\subset} s_2)$
  - Typgleichheit zweier Mengentypen
    $\mathcal{K}_M(\{s_1\}) \overset{\mathrm{T}}{=} \mathcal{K}_M(\{s_1\}) \leftrightarrow s_1 \overset{\mathrm{T}}{=} s_2$
  - Zugriff auf den Datentyp eines Mengenelementes $\mathcal{K}_M(\{s\}).type = s$.
  - Benennung eines Types $\mathcal{K}_M(A).toString =$ 'Set Of'.
  - Reduziere die Präsentation der referenzierten Objekte auf ihre Objektidentifikation (CreateReference) und konvertiere den Mengentyp in einen Referenztyp um.

- Referenz:
  - Erweitere die Präsentation der referenzierten Objekte auf alle Eigenschaften des Objekttyps(CreateSet) und konvertiere den zugrundeliegenden Referenztyp in einen Mengentyp um.
  - Durchschnitt, Vereinigung und Differenz zweier Referenztypen ($\overset{\mathrm{T}}{\bigcup}, \overset{\mathrm{T}}{\bigcap}, \overset{\mathrm{T}}{\setminus}$) mit:
    - $\mathcal{K}_R(\{s_1\}) \overset{\mathrm{T}}{\bigcup} \mathcal{K}_R(\{s_2\}) = \mathcal{K}_R(\{s_1 \overset{\mathrm{T}}{\bigcup} s_2\})$
    - $\mathcal{K}_R(\{s_1\}) \overset{\mathrm{T}}{\bigcap} \mathcal{K}_R(\{s_2\}) = \mathcal{K}_R(\{s_1 \overset{\mathrm{T}}{\bigcap} s_2\})$
    - $\mathcal{K}_R(\{s_1\}) \overset{\mathrm{T}}{\setminus} \mathcal{K}_R(\{s_2\}) = \mathcal{K}_R(\{s_1 \overset{\mathrm{T}}{\setminus} s_2\})$
  - Teilmenge zweier Referenztypen
    $\mathcal{K}_R(\{s_1\}) \overset{\mathrm{T}}{\subset} \mathcal{K}_R(\{s_2\}) \leftrightarrow s_1 \overset{\mathrm{T}}{\subset} s_2)$
  - Typgleichheit zweier Referenztypen
    $\mathcal{K}_R(\{s_1\}) \overset{\mathrm{T}}{=} \mathcal{K}_R(\{s_1\}) \leftrightarrow s_1 \overset{\mathrm{T}}{=} s_2$
  - Zugriff auf den Datentyp eines Referenzelementes $\mathcal{K}_R(\{s\}).type = s$.
  - Benennung eines Types $\mathcal{K}_R(A).toString =$ 'Set Of'.
  - Zuweisung eines Auswahlsatzes (Set).

	$<$	$\leq$	$=$	$\geq$	$>$	$\neq$
$\overset{\text{Integer}}{\Theta}$	1	1	1	1	1	1
$\overset{\text{Double}}{\Theta}$	1	1	1	1	1	1
$\overset{\text{Long}}{\Theta}$	1	1	1	1	1	1
$\overset{\text{Float}}{\Theta}$	1	1	1	1	1	1
$\overset{\text{String}}{\Theta}$	1	1	1	1	1	1
$\overset{\text{Bool}}{\Theta}$			1			1
$\overset{\mathcal{K}_T}{\Theta}$			1			1
$\overset{\mathcal{K}_L}{\Theta}$			1			1
$\overset{\mathcal{K}_M}{\Theta}$	1	1	1	1	1	1
$\overset{\mathcal{K}_{Ref}}{\Theta}$			1			1

Tabelle 3.2: Vergleichsoperationen in Abhängigkeit eines Datentyps

	$+$	$-$	$\backslash$	$*$	$\div$	$\wedge$	$\vee$	$\neg$
$\overset{\text{Integer}}{\Omega}$	1	1	1	1	1			
$\overset{\text{Double}}{\Omega}$	1	1	1	1				
$\overset{\text{Long}}{\Omega}$	1	1	1	1	1			
$\overset{\text{Float}}{\Omega}$	1	1	1	1				
$\overset{\text{String}}{\Omega}$	1							
$\overset{\text{Bool}}{\Omega}$						1	1	1
$\overset{\mathcal{K}_T}{\Omega}$								
$\overset{\mathcal{K}_L}{\Omega}$	1	1						
$\overset{\mathcal{K}_M}{\Omega}$	1	1						
$\overset{\mathcal{K}_{Ref}}{\Omega}$	1							

Tabelle 3.3: Mathematische und logische Operationen in Abhängigkeit eines Datentyps

	$=$	$+=$	$-=$	$\setminus=$	$*=$	$\div=$
$\overset{\text{Integer}}{\Psi}$	1	1	1	1	1	1
$\overset{\text{Double}}{\Psi}$	1	1	1	1	1	
$\overset{\text{Long}}{\Psi}$	1	1	1	1	1	1
$\overset{\text{Float}}{\Psi}$	1	1	1	1	1	
$\overset{\text{String}}{\Psi}$	1	1				
$\overset{\text{Bool}}{\Psi}$	1					
$\overset{\mathcal{K}_T}{\Psi}$	1					
$\overset{\mathcal{K}_L}{\Psi}$	1	1	1			
$\overset{\mathcal{K}_M}{\Psi}$	1	1	1			
$\overset{\mathcal{K}_{Ref}}{\Psi}$	1	1				

Tabelle 3.4: Zuweisungsoperationen in Abhängigkeit eines Datentypes

Generische Operationen

Die generischen Operationen lassen sich entsprechend den folgenden Abbildungen (siehe Abbildung 3.3 und 3.4) nach ihrem Wesen einteilen. Bestimmte Operationen verändern die Objektbeschreibung (Abbildung 3.3), andere (Abbildung 3.4) verändern die Anzahl der Objekte in einem Auswahlsatz.

Die PROJEKTION ist eine Abbildung, die eine Spezialisierung der Objektbeschreibung erzeugt. Die PROJEKTION übernimmt nur den Datentyp, der als Parameter der Operation angegeben wurde. Definiert ist die PROJEKTION als:

$$\mathcal{P}_k(\wp) = \wp' = \{< \sigma, k \overset{\text{T}}{\bigcap} s, z(k \overset{\text{T}}{\bigcap} s) > | < \sigma, s, z(s) > \in \wp \wedge k \in S\}.$$

Das UMBENENNEN ist eine Abbildung, die eine Bezeichnerersetzung vornimmt. Hierzu wird die Komponente des zu ersetzenden Attributes durch seinen Zugriffspfad und der neue Bezeichner angegeben. Der Typ sowie der Zustand der Komponente werden beibehalten.

$$\mathcal{R}_{u \to u\prime}(\wp) = \wp' =$$
$$\{< \sigma, s \overset{\text{T}}{\setminus} \mathcal{K}_T(\{< u, k >\}) \overset{\text{T}}{\bigcup} \mathcal{K}_T(\{< u\prime, k >\}), z(s) > | < \sigma, s, z(s) > \in \wp \wedge u, u\prime \in U \wedge k \in S\}$$

Der VERBUND verändert nicht nur den zugrundeliegenden Typ, sondern auch die Anzahl der Objekte. Der VERBUND kombiniert jedes Objekt des ersten Auswahlsatzes mit jedem Objekt des zweiten Auswahlsatzes. Für die daraus resultierenden Objekte werden neue Objektidentifikatoren vergeben. Der neue Typ ergibt sich aus der Vereinigung beider zugrundeliegender Typen. Formal hat der Verbund folgende Semantik:

$$\mathcal{J}(\wp_1, \wp_2) = \wp' =$$
$$\{< \sigma, s_1 \overset{\text{T}}{\bigcup} (s_2 \overset{\text{T}}{\setminus} s_1), z(s_1 \overset{\text{T}}{\bigcup} (s_2 \overset{\text{T}}{\setminus} s_1)) > | \ \forall < \sigma_1, s_1, z(s_1) > \in \wp_1 \wedge \forall < \sigma_2, s_2, z(s_2) > \in \wp_2\}.$$

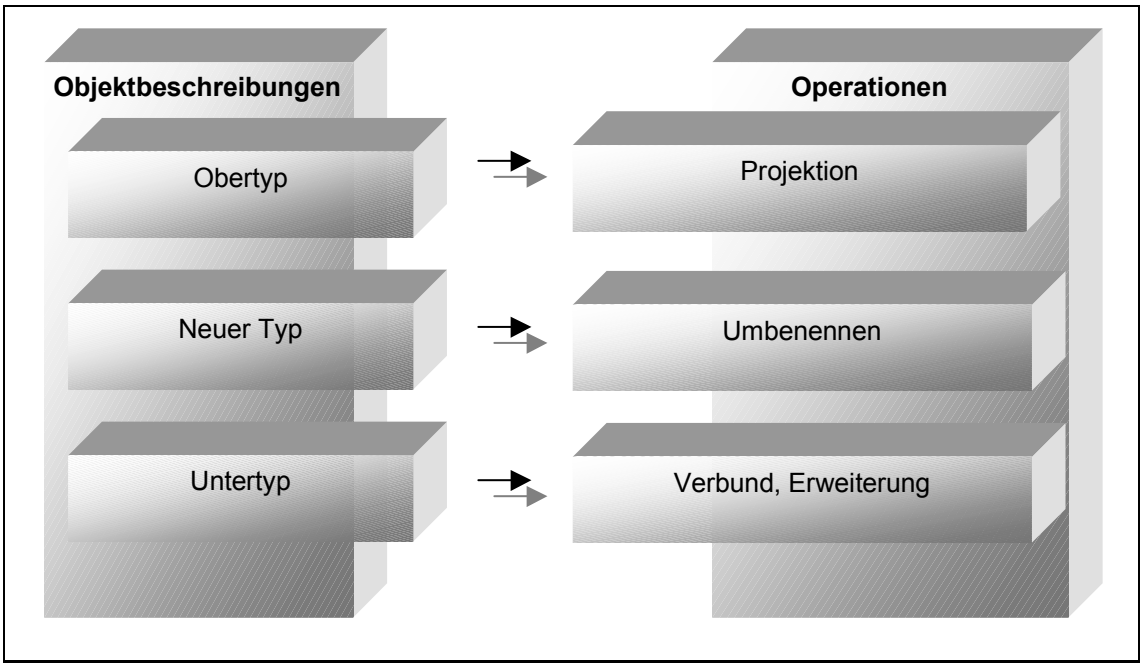


Abbildung 3.3: Operationen für die Manipulation von Objektbeschreibungen

Der Verbund basiert auf zwei Annahmen. Zum einen werden die Attribute vom Objekt o_1 übernommen, wenn beide Objekte gleiche Attributbezeichner benutzen. Zum anderen wird festgelegt, daß $\mathcal{J}(\wp, \{\}) := \{\}$ ist.

Die ERWEITERUNG fügt ein neues Attribut in einer bestehenden Objektbeschreibung ein. Der Wert des Attributes ist abhängig von dem zugehörigen Initialisierer (siehe typspezifische Operationen).

$$\mathcal{X}_{<u,s>}(\wp) =$$
$$\{< \sigma, s \overset{\mathrm{T}}{\bigcup} \mathcal{K}_T(\{< u, s, normal >\}), z(s \overset{\mathrm{T}}{\bigcup} \mathcal{K}_T(\{< u, s, normal >\})) >| < \sigma, s, z(s)) >\in \wp\}$$

Die nächsten Operationen (Abbildung 3.4) verändern nicht den Objekttyp, sondern manipulieren die Elemente eines Auswahlsatzes oder die Attributwerte der zugrundeliegenden Objekte.

Die SELEKTION ordnet die Objekte einem neuen Auswahlsatz zu, welche die angegebene Bedingung erfüllen. Die Identifikation der Objekte bleibt erhalten. Für die Modellierung der Bedingung können zwei mögliche Varianten angegeben werden:

- Bedingungen zwischen zwei Attributen eines Objektes :
  $z(s.u_1) \overset{s.u_1}{\Theta} z(s.u_2)$ mit $\overset{s.u_1}{\Theta}$ siehe Tabelle 3.2 auf Seite 48 und $< u_1, s_1, b >, < u_2, s_2, b >\in s \wedge s.u_1 \overset{\mathrm{T}}{=} s.u_2$.

- Bedingungen zwischen einem Attribut eines Objektes und einer Konstante:
  $z(s.u_1) \overset{s.u_1}{\Theta} c$ mit $\overset{s.u_1}{\Theta}$ siehe Tabelle 3.2 auf Seite 48 und $< u_1, s_1, b >\in s \wedge c \in dom(s.u_1)$.

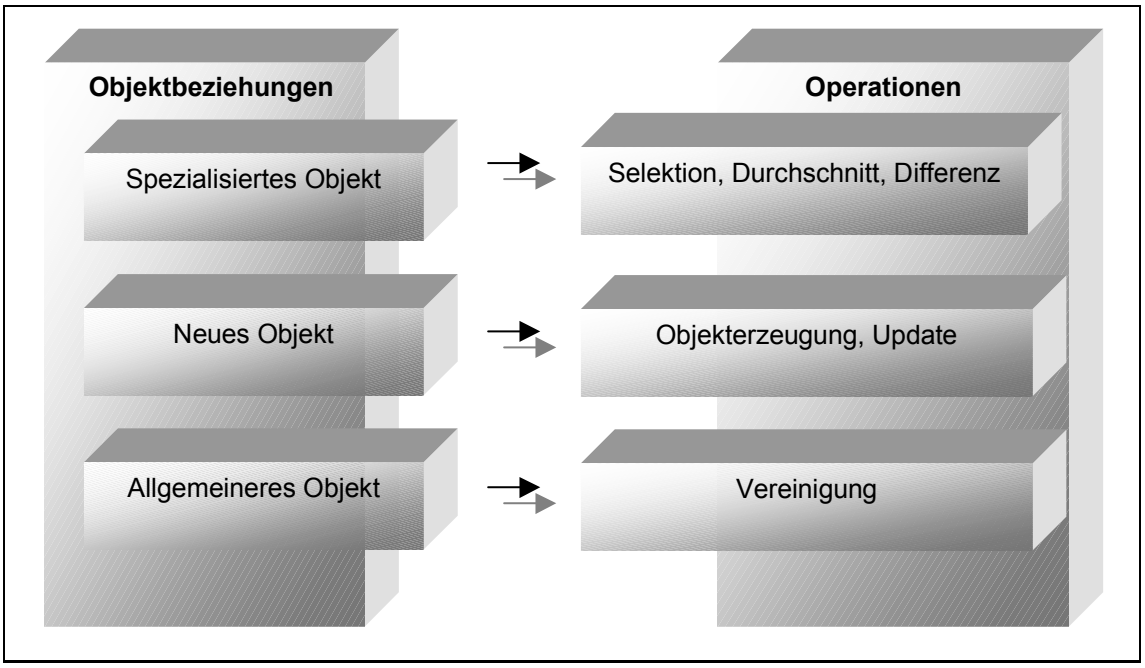


Abbildung 3.4: Operationen für die Manipulation von Auswahlsätzen

Diese beiden Möglichkeiten werden auch Atome der Bedingung genannt. Die SELEKTION unterstützt bei der Angabe die Bildung von Formeln. Eine gültige Formel kann wie folgt gebildet werden:

1. Ein Atom ist eine Formel.

2. Sind B_1 und B_2 Formeln, dann sind $\neg B_1, B_1 \wedge B_2, B_1 \vee B_2, (B_1)$ gültige Formeln.

Die SELEKTION wird demnach wie folgt definiert:

$\mathcal{S}_B(\wp) = \wp' = \{o \mid o \in \wp \wedge B\}$
mit B eine gültige und auswertbare Formel für das Objekt o.

Der DURCHSCHNITT wählt nur die Objekte aus, die in den beiden angegebenen Auswahlsätzen vorhanden sind. Seine Spezifikation hat folgendes Erscheinungsbild:

$\mathcal{A}(\wp_1, \wp_2) = \wp' = \{< \sigma, s, z(s) > | < \sigma, s, z(s) > \in \wp_1 \wedge < \sigma, s, z\prime(s) > \in \wp_2\}$.

Die DIFFERENZ ergibt sich aus allen Objekten, die sich im ersten Auswahlsatz befinden, aber nicht im zweiten. Formal hat die Differenz folgende Spezifikation:

$\mathcal{M}(\wp_1, \wp_2) = \wp' = \{< \sigma, s, z(s) > | < \sigma, s, z(s) > \in \wp_1 \wedge < \sigma, s, z\prime(s) > \notin \wp_2\}$

Die Operation OBJEKTERZEUGUNG fügt ein neues Objekt in den angegebenen Auswahlsatz ein. Der Objekttyp wird durch den zugrundeliegenden Datentyp der Objekte des Auswahlsatzes

spezifiziert. Für die Zustandsfunktion gilt das gleiche wie für die weiter oben spezifizierte Methode APPEND. Die Attributwerte ergeben sich aus den voreingestellten Werten in den Initialisierer.

Mit der UPDATE-Operation ist die Zuweisung eines neuen Zustandes möglich. Hierzu kann ein geordnetes Tripel angegeben werden, welches den Zuweisungsoperator Ψ (siehe Tabelle 3.4 auf Seite 49), den Attributnamen, dessen Attributwert manipuliert werden soll, sowie den neuen angegebenen Wert enthält.
$z(s.u) \overset{s.u}{\Psi} v$ mit $v \in dom(s.u)$

Der neue Wert kann entweder als eine Konstante ($v = c$), als ein Attribut des Objektes ($v = z(s.u)$) oder durch den Rückgabewert einer Funktion ($v = f(\{v_i\})\, i \in \mathbb{N}$) angegeben werden. Diese drei Möglichkeiten nennt man auch die Atome des rechten Terms einer Zuweisung. Zusätzlich unterstützt die UPDATE-Operation die Bildung von Ausdrücken für die Angabe des rechten Terms. Ein gültiger Ausdruck unterliegt dabei folgenden Axiomen:

- Ein Atom ist ein gültiger Ausdruck.

- Sind a_1 und a_2 gültige Ausdrücke und $a_1, a_2 \in dom(s)$ dann sind $a_1 \overset{s}{\Omega} a_2$ gültige Ausdrücke.

Die UPDATE-Operation erhält als Parameter einen Auswahlsatz an Objekten und eine Menge von geordneten Paaren. Für jedes Objekt innerhalb des Auswahlsatzes wird die Wertersetzung durchgeführt:

$$\mathcal{U}_{u\overset{s.u}{\Psi}a}(\wp) = \wp' = \{o \mid o \in \wp \wedge s.u \in S \wedge a \in dom(s.u) : z(s.u) \overset{s.u}{\Psi} a\}.$$

Die VEREINIGUNG erlaubt das Zusammenfassen von Objekten aus zwei Auswahlsätzen zu einem Auswahlsatz. Die Vereinigung wird wie folgt definiert:

$$\mathcal{N}(\wp_1, \wp_2) = \wp' = \{< \sigma, s, z(s) > | < \sigma, s, z(s) > \in \wp_1 \vee (< \sigma, s, z(s) > \in \wp_2 \wedge < \sigma, s, z\prime(s) > \notin \wp_1)\}.$$

Einleuchtend ist es, daß die vorgestellten Operationen vollständig in der Objektbeschreibung des Auswahlsatzes vorhanden sein müssen. Die Implementierung der Operationen in der Objektbeschreibung des Auswahlsatzes muß auf die Implementierung der verwendeten Objekte zurückgreifen. Aus diesem Grund ist es zweckmäßig, daß die Implementierungen der Operationen in den Objekten sich aus den Implementierungen in den einzelnen Datentypen zusammensetzen.

Demnach müssen die entsprechenden Datentypen, die für die Bildung von zusammengesetzten Datentypen verwendet werden, die notwendige Implementierung für diese Operationen polymorph bereitstellen. Da nicht für jede Operation eine sinnvolle Anwendung existiert, müssen die gültigen Operationen definiert werden. Tabelle 3.5 faßt die gültigen Operationen für die entsprechenden zusammengesetzten Datentypen zusammen.

	$\mathcal{K}_T$	$\mathcal{K}_L$	$\mathcal{K}_M$	$\mathcal{K}_{Ref}$
PROJECTION	x	x	x	
UMBENENNUNG	x	x	x	
VERBUND		x	x	
ERWEITERUNG	x	x	x	
SELECTION		x	x	
DURCHSCHNITT		x	x	
DIFFERENZ		x	x	
INSERT		x	x	
UPDATE	x	x	x	
VEREINIGUNG		x	x	
CREATESET				x
CREATEREFERENCE		x	x	
HASELEMENT		x	x	

Tabelle 3.5: gültige Operationen komplexer Metaklassen

3.2.3 Aktionen für die Wahrung der Konsistenz

Das Ziel, welches diese Gruppe von Aktionen ermöglichen soll, ist es, die einzelnen Bearbeiter über den Konsistenzzustand[3] der einzelnen Planungsunterlagen zu informieren und bei der Aktualisierung der einzelnen Planungsunterlagen den Bearbeiter zu unterstützen.

Die Planungsunterlagen lassen sich in verschiedene Zustände einteilen. Da nicht zu jedem Zeitpunkt alle Planungsunterlagen zueinander konsistent oder inkonsistent sind, reichen diese beiden nicht aus. Zweckmäßig ist es, vier verschiedene Zustände für Planungsunterlagen einzuführen.

Eine Planungsunterlage ist:

1. **stark konsistent,**
   wenn alle Objekte der Planungsunterlagen sowie die verwendeten Objekte in den anderen Planungsunterlagen in konsistenter Form vorliegen.
2. **schwach konsistent,**
   wenn mindestens ein Objekt der Planungsunterlage in einer inkonsistenten Form vorliegt, sowie die verwendeten Objekte in den anderen Planungsunterlagen in konsistenter Form vorliegen.
3. **schwach inkonsistent,**
   wenn alle Objekte der Planungsunterlage in einer konsistenten Form vorliegen und wenn

[3] Konsistenz in diesem Zusammenhang bedeutet nicht, daß die einzelnen Planungsunterlagen zueinander korrekt sind. Für die Korrektheit der Planungsunterlage ist der Anwender verantwortlich. Konsistenz in diesem Zusammenhang bedeutet, daß die Bearbeiter alle vom Integrationsmodul generierten Nachrichten verarbeitet haben.

es mindestens ein verwendetes Objekt in einer anderen Planungsunterlage gibt, das in inkonsistenter Form vorliegt.

4. **stark inkonsistent,**
   wenn mindestens ein Objekt der Planungsunterlage sowie ein verwendetes Objekt in einer anderen Planungsunterlage in inkonsistenter Form vorliegen.

Sind alle Planungsunterlagen stark konsistent, dann ist das Gesamtmodell konsistent.

Das zweite Teilziel muß die Arbeitsweise der Anwender beachten. Laut dieser Arbeitsweise müssen die Anwender zu bestimmten Zeitpunkten ihre Planungsunterlagen untereinander abstimmen. Dazu müssen Veränderungen aus den zugrundeliegenden Planungsunterlagen erkannt werden und die Planungsunterlagen, die diese Informationen verwenden, entsprechend der aufgetretenen Veränderungen angepaßt werden.

Für eine effiziente Abwicklung der Aktualisierung der Planungsunterlagen, die durch den jeweiligen Bearbeiter durchzuführen ist (vergleiche dazu Abschnitt 1.3.2 auf Seite 19), sind folgende Informationen hilfreich:

1. die veränderten Objekte der aktualisierten Planungsunterlage, klassifiziert nach der Art der Veränderung,

2. die Objekte in der betrachteten Planungsunterlage, die von den Veränderungen der Objekte der aktualisierten Planungsunterlage betroffen sind.

Demnach basiert die Wahrung der Konsistenz auf drei Prozessen:

- Ermittle die Veränderungen einer bestimmten Planungsunterlage (Σ_i) auf der Basis veränderter Objektinformationen.

- Für alle anderen Planungsunterlagen $\Sigma_0, .., \Sigma_{i-1}, \Sigma_{i+1}, .., \Sigma_k$, ermittle die Objekte, die durch die veränderten Obektinformationen in Σ_i aktualisiert werden müssen.

- Aktualisiere die Objekte in den Planungsunterlagen $\Sigma_0, .., \Sigma_{i-1}, \Sigma_{i+1}, .., \Sigma_k$ entsprechend den Veränderungen in Σ_i.

Die ersten beiden Prozesse können von entsprechenden Softwarewerkzeugen ohne Anwenderinteraktionen ermöglicht werden. Der Anwender muß den letzten Prozeß realisieren.

Die möglichen Veränderungen können durch folgende Überarbeitungen auftreten:

- Neue Objekte wurden in die Planungsunterlage eingefügt.

- Bestehende Objekte wurden aus der Planungsunterlage entfernt.

- Bestehende Objekte wurden modifiziert.

Die Veränderungen von Objektinformation einer Planungsunterlage können erkannt werden, indem die Objekte der letzten Freigabe einer Planungsunterlage mit denen der neuen, überarbeiteten Planungsunterlage verglichen werden. Wird die Menge aller Objekte der alten Planungsunterlage der Applikation k mit Σ_k und die Menge aller Objekte der gleichen Planungsunterlage, die jedoch im überarbeiteten Zustand vorliegt mit Σ_k^* bezeichnet, so enthalten die Mengen O_k^D, O_k^I und O_k^M die Identifikatoren:

- aller gelöschten Objekte
  $O_k^D = \{\sigma \mid < \sigma, s, z(s) > \in \Sigma_k \wedge < \sigma, s, z^*(s) > \notin \Sigma_k^*\}$,
- aller eingefügten Objekte
  $O_k^I = \{\sigma \mid < \sigma, s, z(s) > \notin \Sigma_k \wedge < \sigma, s, z^*(s) > \in \Sigma_k^*\}$,
- aller modifizierten Objekte
  $O_k^M = \{\sigma \mid < \sigma, s, z(s) > \in \Sigma_k \wedge < \sigma, s, z^*(s) > \in \Sigma_k^* \wedge z(s) \neq z^*(s)\}$

aus den angegebenen Mengenrelationen.

Die Mengen α_k^D und α_k^M, die die Identifikatoren der Objekte beinhalten, welche in einer anderen Planungsunterlage auf Grund der aufgetretenen Veränderungen in der Planungsunterlage Σ_k aktualisiert werden müssen, werden aus den Mengen O_k^M, O_k^D und aus der Relation ρ gebildet. Die Relation beschreibt Abhängigkeiten zwischen zwei beliebigen Objekten. Die Relation besagt, daß zwei Objekte $< \sigma_1, s_1, z(s_1) >, < \sigma_2, s_2, z(s_2) >$ mit $\sigma_1 \rho \sigma_2$, voneinander abhängig sind. Das heißt, tritt eine Veränderung am Objekte σ_1 auf, dann muß das Objekt σ_2 auf seine Richtigkeit im Gesamtmodell überprüft werden. Die Bildungsvorschriften für die Mengen α_k^D und α_k^M lauten:

$\alpha_k^D = \{\sigma \mid < \sigma, z, (s) > \in \bigcup_i \Sigma_i \backslash \Sigma_k \wedge \sigma_1 \in O_k^D \wedge \sigma_1 \rho \sigma\}$
$\alpha_k^M = \{\sigma \mid < \sigma, z, (s) > \in \bigcup_i \Sigma_i \backslash \Sigma_k \wedge \sigma_1 \in O_k^M \wedge \sigma_1 \rho \sigma\}$.

Diese Mengen lassen sich in Äquivalenzklassen zerlegen, wobei die Zugehörigkeit eines Objektes zu einer Planungsunterlage das Kriterium der Zerlegung darstellt. Jede Planungsunterlage kann somit über drei Mengen verfügen. Eine Menge, die alle Objekte der betrachteten Planungsunterlage beinhaltet und auf Grund eines Löschvorganges eines Objektes in einer anderen Planungsunterlage aktualisiert werden muß. Eine Menge von Objekten, die alle Objekte der betrachteten Planungsunterlage beinhaltet und auf Grund einer Manipulation eines Objektes in einer anderen Planungsunterlage aktualisert werden muß, sowie eine Menge von Objekten, die in den anderen Planungsunterlagen neu eingefügt wurden.

Weitere notwendige Aktionen beschäftigen sich mit dem Ziel, den Bearbeitern die ausschließlich gewünschten Informationen für die Bearbeitung des Prozesses bereitzustellen. Die Aktionen basieren auf der Erzeugung und Bearbeitung der Elemente der Relation ρ sowie auf der Konfiguration eines spezifischen, von der betrachteten Planungsunterlage abhängigen Filters, der entsprechend den konfigurierten Einstellungen nur die Objekte dem Bearbeiter präsentiert, die nach einer Einschätzung des Bearbeiters eine relevante Rolle in der vom Bearbeiter betrachteten Planungsunterlage spielen.

Die Aktionen sind notwendig, um den Bearbeitern nicht durch zu umfangreiche Informationen über die Veränderungen in den anderen Planungsunterlagen zu überfordern. Ein Überschuß an unerwünschten Informationen führt zu einer Beeinträchtigung der angestrebten Effizienz.

Die Manipulation der Relation ρ stützt sich auf das Hinzufügen bzw. Löschen von Elementen.

- automatisches Hinzufügen von Elementen

  Automatisch werden neue Elemente während der "Übernahme von Information"erzeugt. Werden Objektinformationen aus bestehenden Planungsunterlagen selektiert, die danach für die Erzeugung eines neuen Objektes verwendet werden, so werden automatisch die beiden beteiligten Objekte durch ihre Identifikation in Relation gesetzt. Die Grundlage hierfür ist, daß davon ausgegangen werden kann, daß eine Manipulation des Ausgangsobjektes eine Reaktion im Zielobjekt auslöst. Das heißt, die beiden beteiligten Objekte sind abhängig.

- automatisches Löschen von Elementen

  Wie das Hinzufügen eines neuen Elementes gibt es eine sinnvolle Anwendung, Elemente der Relation ρ automatisch zu entfernen. Sie ergibt sich, sobald ein Objekt gelöscht wird, welches Element des Definitions- oder Wertebereiches der Relation ρ ist. Das Löschen eines Objektes bedingt das Löschen aller Tupel der Relation, die gelöschte Objekte benutzen.

  Das automatische Entfernen von Tupeln birgt noch eine Gefahrenquelle. Die Gefahrenquelle ergibt sich, sobald ein gelöschtes Objekt Element des Definitionsbereiches der Relation ist. Die Aktion muß sicherstellen, daß alle Objekte des Wertebereiches ermittelt werden, die von einem gelöschten Objekt abhängen. Erst danach können die entsprechenden Tupel entfernt werden.

Die automatische Bearbeitung der Relation ρ operiert immer auf zwei Objekten. Im Gegensatz dazu kann die Anzahl bei den manuellen Aktionen, die durch den Bearbeiter gestartet werden, variieren. Im Hinblick einer einheitlichen Modellierung der Relation ρ sollten die manuellen Aktionen auch immer nur zwei Objekte in Beziehung setzen. Zweckmäßig ist es jedoch, die Angabe von Platzhaltern während der Eingabe der Aktion in der Benutzeroberfläche zu unterstützen. Das sind:

- alle Objekte,
- alle Objekte eines Modells,
- alle Objekte eines Modells und einer Klasse,
- ein bestimmtes Objekt.

Eine weitere Einschränkung ergibt sich aus der Frage: Welche Objekte kann ein Bearbeiter in Beziehung setzen? Nicht akzeptabel ist es, die Menge der möglichen Objekte des Defintionsbereiches der Relation einzuschränken. Akzeptabel ist es, nur die Objekte für den Wertebereich zuzulassen, die Element der Planungsunterlage sind, die von dem jeweiligen Bearbeiter bearbeitet wird.

- manuelles Hinzufügen von Elementen

  Das manuelle Hinzufügen von Tupeln muß im Gegensatz zum automatisierten Vorgehen beachten, daß ein Bearbeiter über die Manipulation von Objekten informiert werden möchte, obwohl einer Veränderung der Attributbelegung von Objekten, die in fremden Planungsunterlagen eingebettet wurden, keine direkte Veränderung der Attributbelegung der eigenen

Objekte folgt. Demnach muß die Aktion die Angabe von real existierenden Objekten sowie die Angabe eines Objektes (Stellvertreterobjekt), welches nicht Element des Datenraums der Planungsunterlage ist aber diese Planungsunterlage im Nachrichtenkonzept vertritt, unterstützen.

- manuelles Löschen von Elementen

  Das manuelle Löschen von Tupeln der Relation ρ kann nur für die Tupel erfolgen, die manuell vom gleichen Bearbeiter erzeugt wurden. Das Löschen von Tupeln führt zum sofortigen Bereinigen der Mengen, welche die Objekte beinhalten, die in der jeweiligen Planungsunterlage aktualisiert werden sollten. Das heißt, ein Objekt o_i der Planungsunterlage, das durch eine Veränderung eines Objektes o_k in einer anderen Planungsunterlage aktualisiert werden sollte, wird mit dem Löschen des Tupels $< o_i, o_k > \in \rho$ auch nicht weiter in der Menge der zu aktualisierenden Objekte der entsprechenden Planungsunterlage geführt.

Eine weitere Aktion in diesem Zusammenhang ermöglicht das Bilden von verschiedenen Sichten auf die Objektmengen, die aktualisiert werden sollen. Jeder Bearbeiter kann eine beliebige Anzahl von Filter definieren. Demnach lassen sich folgende Merkmale für die Zerlegung der Objekte, die in der betrachteten Planungsunterlage aktualisiert werden müssen, angeben:

- Zerlegung entsprechend der Generierung des Tupels (automatisch oder manuell),
- Zerlegung entsprechend der Art der Veränderung der Objekte des Definitionsbereiches (eingefügt, modifiziert, gelöscht),
- Zerlegung entsprechend der Zugehörigkeit der Objekte des Definitionsbereiches zu einer ausgewählten Planungsunterlage und entsprechenden Ojektbeschreibungen,
- Zerlegung entsprechend der real existierenden Objekte in der Planungsunterlage und dem Stellvertreter-Objekt von der Planungsunterlage.

Die letzte Aktion ermöglicht die Darstellung der Relation ρ in Abhängigkeit eines ausgewählten Objektes, das in der betrachteten Planungsunterlage aktualisiert werden soll. Die Darstellung soll folgende zwei Aspekte dem Bearbeiter erläutern:

- Von welchen Objekten ist das gewählte Objekt abhängig, und welche Objekte verwendet das gewählte Objekt?
- Welche Objekte sind in einem inkonsistenten Zustand?

3.3 Architektur der Softwarekomponenten

Faßt man das zuvor Beschriebene noch einmal zusammen, so muß die verwendete Architektur folgende Aspekte berücksichtigen:

- Integration von Planungsunterlagen unterschiedlichster Applikationen in einem Container,

- Bereitstellung eines Zugriffes auf die jeweils aktuellste und freigegebene Version einer Planungsunterlage unabhängig von einem Zeitpunkt,
- Ermöglichung des parallelen Arbeitens an den Planungsunterlagen,
- Unterstützung der internen Verfeinerung von Planungsunterlagen und Aktualisierung dieser in bestimmten Abständen,
- Benutzung einer Planungsunterlage von unterschiedlichen Anwendern für verschiedene Verwendungszwecke gleichzeitig,
- Bereitstellen von Aktionen im Container für die Realisierung des Integrationsprozesses, wie:
  - Zugriff auf die modellierten Daten in den integrierten Planungsunterlagen für die Erfassung von Objekttyp und Zustand,
  - Manipulation der modellierten Daten,
  - Erzeugen von Instanzen in einer ausgewählten Planungsunterlage unter der Berücksichtigung ermittelter Daten,
  - Speicherung von Objektzuständen und deren Beziehungen untereinander,
  - Wahrung der Konsistenz zwischen den Planungsunterlagen,
- Aktualisierung von Planungsunterlagen

Die Realisierung der notwendigen Aktionen und Daten des Integrationsprozesses in einer separaten Softwarekomponente, dem Integrationsmodul, ist zweckmäßig, da die Aufgaben der Integration nicht von der verwendeten Applikation abhängen. Das Integrationsmodul integriert somit die verwendeten Planungsunterlagen und implementiert ein Nachrichtenkonzept, daß die Wahrung der Konsistenz zwischen den einzelnen Planungsunterlagen gewährleistet, sowie eine assoziative Abfragesprache, die den Anwendern eine Eingabeform zur Verfügung stellt, mit der sie typunabhängig und individuell die übernommenen Informationen bearbeiten können.

Die aktive Integration des Anwenders bedingt die semantische Interpretation der von den Modellen verwendeten Objekttypen durch den Anwender. Dies ermöglicht den Verzicht auf ein semantisches Objektmodell. Der Verzicht auf ein semantisches Objektmodell erfordert jedoch, daß das Integrationsmodul die verwendeten Objekttypen von den betrachteten Applikationen erfragen und diese Objekttypen wie die Zustände der ausgewählten Objekte dynamisch im Integrationsmodul nachbauen kann. Demnach benötigt der Integrationsansatz eine Technologie, mit der das Integrationsmodul mit den einzelnen Applikationen kommunizieren kann. Die Technologie, die in dieser Arbeit für die Implementierung des Prototyps verwendet wurde, ist die ActiveX-Automation sowie eine Reihe selbst definierter COM-Interfaces.

Die einzelnen Komponenten des Integrationsansatzes sowie die verwendeten Technologien sind in der Abbildung 3.5 dargestellt.

Der Anwender steuert das Integrationsmodul sowie die einzelnen Applikationen mit ihren dargestellten Modellen in der Benutzeroberfläche. Das Integrationsmodul kopiert die ausgewählten Objekte in den einzelnen Applikationen in den persistenten Datenraum des Integrationsmoduls. Somit liegen die Objekte zwar redundant im Integrationsmodul und in den einzelnen Modellen

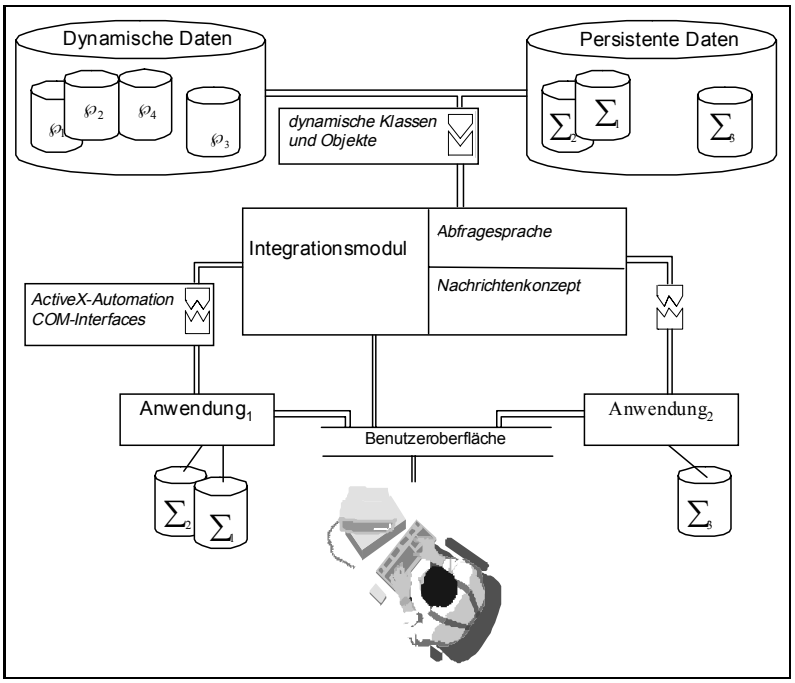


Abbildung 3.5: Architektur der Softwarekomponenten

der Applikationen vor, sie verlieren jedoch nicht die Zuordnung zueinander, da die Zuordnung mittels der Objektidentifikationen weiterhin gegeben ist.

Die Verwendung einer Client-Server Softwarearchitektur ist zweckmäßig, da die geforderten Funktionalitäten des Integrationskonzeptes für jede Applikation die gleichen sind und somit nur einmal bereitgestellt werden müssen. Die Anwendungen fungieren als Server und stellen dem Integrationsmodul nur die notwendigen Kommunikationsmethoden zur Verfügung.

Eine weitere Entscheidung befaßt sich mit der Frage, welche Objekte persistent und welche Objekte temporär im Integrationsmodul abzulegen sind. Alle Objekte, die aus bestehenden Planungsunterlagen extrahiert sowie durch die Anwendung der Aktionen (siehe Abschnitt 3.2 auf Seite 39) gewonnen werden, persistent abzulegen, würde mit Sicherheit zu Performanceproblemen in der Datenverwaltung des Integrationsmoduls führen.

Soll die Menge an persistenten Informationen minimiert werden, so empfiehlt es sich, die Grenze zwischen den persistenten und temporären Objekten an Hand des Verwendungszweckes eines Objektes anzugeben. Die Integrationsaufgabe 'Wahrung der Konsistenz' erfordert Persistenz für alle Objekte, die zu überwachen sind. Wird ein Objekt im Integrationsprozeß 'Übernahme von Informationen' verwendet, dann wird eine temporäre Kopie erzeugt, die der Anwender für die weitere Bearbeitung verwenden kann. Diese Aufteilung ermöglicht die Bearbeitung von ausgewählten Objekten unabhängig von den Objekten in den speziellen Planungsunterlagen. Somit können alle Bearbeiter auf die gleichen persistenten Objekte parallel zugreifen, ohne sich gegenseitig zu behindern.

3.3.1 Auswahl und Einsatz der Kommunikationsplattform COM

Das Integrationsmodul integriert die einzelnen Planungsunterlagen und ermöglicht über spezielle Schnittstellen (siehe Abschnitt 4.3 auf Seite 112) den Austausch von Informationen. Mittels dieser Schnittstellen übernimmt das Integrationsmodul die einzelnen Objekte aus den einzelnen Planungsunterlagen und stellt diese Objekte dem Anwender im Integrationsmodul zur Verfügung. Für die Implementierung der Kommunikation zwischen den einzelnen Applikationen und dem Integrationsmodul wurde im Prototyp die COM-Technologie (component object model) verwendet.

COM stellt für verschiedene Dienste die Basistechnologie dar. Ein Dienst z.B. ist die Automation. Die Automation-Schnittstelle ist die Grundlage für die Programmiersprache Visual Basic. In zunehmendem Maße benutzen kommerzielle Softwarehersteller die Spezifikation dieser Schnittstelle, um den Anwendern eine Möglichkeit einzuräumen, die Programmfunktionalität des Softwareproduktes zu erweitern um somit noch effizienter mit dem Softwareprodukt arbeiten zu können. Diese Erweiterung wird üblicherweise mit dem Namen Visual Basics for Applications (VBA) bezeichnet. Die zunehmende Verbreitung von VBA bei den Softwareherstellern war ausschlaggebend für den Einsatz der COM-Technologie bei dem hier vorgestellten Ansatz, da die Softwareprodukte bereits über die Automation-Schnittstelle für die applikationsspezifischen Objekte verfügen.

Ein weiterer Grund, der für den Einsatz von COM als Kommunikationsbasis spricht, ist die binäre Unabhängigkeit zwischen Server und Client. Laut [Box98] ist COM die einzige Kommunikationsplattform, welche die Grundlage für die unabhängige Implementierung zwischen Client und Server garantiert. Demnach ist es unter COM möglich, einen Server zu ersetzen, ohne eine erneute Compilierung des Clients durchzuführen. Diese Möglichkeit ist für den Integrationsansatz unbedingt notwendig. Es wäre fatal, jedesmal eine neue Compilierung des Clients zu fordern, wenn beispielsweise eine neue Applikation, die von einem Anwender verwendet wird, in den Integrationsprozeß zu integrieren ist.

COM[4] stellt die Basis für verschiedene Dienste dar, um eine Kommunikation zwischen zwei Applikationen aufzubauen. Die Applikation, die einen Dienst bereitstellt, wird als Server bezeichnet, die Applikation die einen bereitgestellten Dienst benutzt, wird als Client bezeichnet. Das prinzipielle Vorgehen eines Clients, wenn er einen Dienst des Servers verwenden will, ist:

1. Erzeuge einen Zugriff auf eine Instanz des Servers.
2. Ermittle einen Zugriff auf den bereitgestellten Dienst.
3. Aktiviere den Dienst.

Die Grundvoraussetzung hierfür ist, daß der Client den gewünschten Server identifizieren kann und daß er den Umfang, die Semantik sowie die Syntax der angebotenen Dienste des Servers kennt.

Für die Identifizierung des Servers wird ein 128-Bit großer persistenter Schlüssel verwendet. Die Generierung des Schlüssels ist Aufgabe des Servers, der üblicherweise hierfür ein Tool, welches die Eineindeutigkeit des Schlüssels garantiert, verwendet.

[4] Detaillierte Hintergrundinformationen über COM gibt [Box98]. Die Spezifikation von OLE und OLE-Automation ist in [OLE 2] und [Brock95] enthalten.

Für die Spezifikation der angebotenen Dienste werden abstrakte Klassen verwendet, die auch als Schnittstelle (engl. Interface) bezeichnet werden. Jedes Interface besteht ausschließlich aus pur virtuellen Methodendeklarationen. Diese Interface-Deklarationen werden vom Server und vom Client eingebunden, wobei nur der Server die Methoden des Interface' implementiert.

Die Codierung des Clients kann somit die eingebundenen Klassendeklarationen verwenden und die entsprechenden Methoden aufrufen. Da diese Methoden pur virtuell sind, wird während der Erzeugung des Clientcodes nicht die Implementierung der Methoden eingebunden. Erst zur Laufzeit, wenn der Client einen gültigen Verweis zum Server aufgebaut hat, erfolgt die Zuordnung der Anweisungen, die die aufgerufenen Methoden im entsprechenden Server verkörpern.

Die Grundlage für das Benutzen von Diensten, ohne hierfür einen speziellen Server und Client zu betrachten, liegt in der Festlegung von Interface-Beschreibungen. Die Microsoft-Betriebssysteme legen eine Reihe von Interface-Beschreibungen fest, um eine gezielte Kommunikation zwischen Server und Client für bestimmte Aufgaben aufzubauen.

Eine Aufgabe, die Windows-Applikationen standardmäßig unterstützen, ist das Einbetten einer Präsentation, die mit einem Server erstellt wurde, in die Präsentation des Clients. Der dabei notwendige Kommunikationsaustausch umfaßt die Darstellung, die Aktivierung und die persistente Datenhaltung der eingebetteten Präsentation. Diese Technologie wird mit dem Namen OLE (object linking and embedding) bezeichnet. Im Integrationsansatz wird diese Technologie benutzt, um die verschiedenen Planungsunterlagen in einen Windows-Container einzubetten, wobei die grafische Präsentation der Planungsunterlage im Container unberücksichtigt bleibt.

Eine weitere Aufgabe befaßt sich mit dem Zugriff auf die Daten eines Objektes einer eingebetteten Planungsunterlage. Der hier erforderliche Kommunikationsaustausch umfaßt die Ermittlung der Spezifikation von Objekten sowie den Zugriff auf die zur Verfügung stehenden Properties in Abhängigkeit eines Objektes. Dieser Aspekt wird als OLE-Automation bezeichnet. Da die OLE-Automation eine Schlüsselrolle in diesem Integrationsansatz spielt, soll sie noch etwas detaillierter erläutert werden.

Das Verständnis der OLE-Automation erfordert die Erweiterung des Begriffes Server. Bei der Automation können alle Instanzen von Klassen Server sein, sobald diese ein bestimmtes Interface zur Verfügung stellen. Das betrifft ein einzelnes Objekt, einen Container für eine Menge von Objekten bis hin zur eigentlichen Applikation, die auch den Einstiegspunkt der Automation bildet und somit als Wurzel für alle anderen Objekten betrachtet werden kann.

Neben der Implementierung des vorgegebenen Interface' beinhaltet jedes Objekt eine Menge von Attributen und Methoden. Will nun der Client eine bestimmte Methode des Server-Objektes starten, so benutzt er einen ausgezeichneten Dienst des Interface', dessen Parameter den Methodenaufruf der angegebenen Aktion wiedergibt. Die Implementierung des Dienstes auf der Serverseite stellt sicher, daß die entsprechende Methode des Objektes mit den angegebenen Parametern aufgerufen wird. Das Resultat der Methode wird nach dem Beenden der Methode durch die entsprechende Belegung der Parameter im Dienst dem Client zur Verfügung gestellt. Der Zugriff auf den Wert eines Attributes erfolgt nach dem gleichen Schema.

Eine Frage bleibt: Woher bekommt der Client die Information, welche zusätzlichen Attribute und Methoden ein bestimmtes Objekt unterstützt? Hierzu benutzt der Client eine ausgezeichnete Datei, die TypeLibrary, die mit dem Server verbunden ist. Diese Datei ist hierarchisch aufgebaut und erlaubt dem Client eine Analyse, beginnend bei den verfügbaren Serverobjekten,

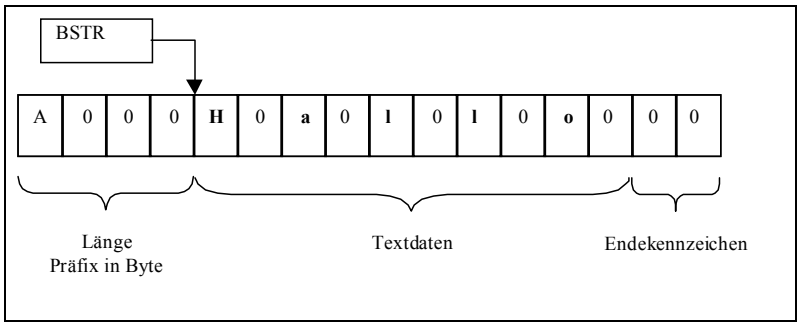


Abbildung 3.6: Zeichenkette "Hallo" als BSTR

über die von den Objekten verwendeten Interfaces bis hin zur Dokumentation einer Attribut- oder Methodendeklaration.

3.3.2 Datentypen

Entsprechend der Strukturbeschreibung müssen die verwendeten atomaren Datentypen sowie die eingesetzten Typkonstruktionen festgelegt werden. Im Abschnitt 2.1.2 auf Seite 27 wurde hergeleitet, daß der Entwickler für die Modellierung der Datenstrukturen eine Menge von atomaren Datentypen benutzt, die er mittels Typkonstruktionen und bereits bekannten Datentypen zu neuen Datentypen kombinieren kann.

Die vom Entwickler verwendeten Datentypen werden in die Datentypen konvertiert, die von der Kommunikationskomponente verwendet werden. Im Abschnitt 3.5 auf Seite 59 wurde schon erwähnt, daß für den Prototyp die Kommunikationskomponente COM eingesetzt wird.

Speziell für den Zugriff auf die einzelnen Eigenschaften der Objekte wird die ActiveX-Automation eingesetzt. Die hierfür möglichen Datentypen müssen als gegeben betrachtet werden. Die Beschreibung aller COM-Interfaces (einschließlich ActiveX Automation) erfolgt in IDL (Interface Definition Language). Die Tabelle 3.6 listet die in Frage kommenden Datentypen mit den entsprechenden Ausprägungen in den Programmiersprachen auf.

Die IDL-Datentypen erlauben eine sprach- und plattformunabhängige Art, Datentypen in den Programmiersprachen zu beschreiben. Die Datentypen, die für die Beschreibung von gebrochenen und ganzen Zahlen benutzt werden, bedürfen kaum einer weiteren Erläuterung. Interessant sind die Datentypen, die für die Beschreibung von Zeichenketten, Feldern und selbst definierten Datentypen benutzt sowie für die Definition von Objektbeziehungen verwendet werden.

Für die Beschreibung von Zeichenketten steht der Datentyp BSTR zur Verfügung. Dieser Datentyp setzt sich aus drei Teilen zusammen. Der erste Teil (Präfix) beinhaltet die Anzahl der Bytes, die für die Zeichenkette verwendet werden und ist 4 Byte lang. Der zweite Teil beinhaltet die Zeichenkette. Jedes Zeichen wird in dem 16 Bit langen Unicode abgebildet. Der dritte Teil beinhaltet die Endekennung (zwei Bytes) der Zeichenkette. Diese zwei Bytes werden im Präfix nicht mitgezählt. Die Abbildung 3.6 zeigt die Darstellung der Zeichenkette "Hallo"als BSTR.

Sprache	IDL	Microsoft C++	Visual Basic	Microsoft Java
Basistypen	boolean	unsigned char	nicht unterstützt	char
	byte	unsigned char	nicht unterstützt	char
	small	char	nicht unterstützt	char
	short	short	Integer	short
	long	long	Long	int
	Hyper	__int64	nicht unterstützt	long
	float	float	Single	float
	Double	double	Double	double
	char	unsigned char	nicht unterstützt	char
	Enum	enum	Enum	int
	Schnittstellenzeiger	Schnittstellenzeiger	Schnittstellenzeiger	Schnittstellenzeiger
Erweiterte Typen	VARIANT	VARIANT	Variant	ms.com.Variant
	BSTR	BSTR	String	java.lang.String
	VARIANT_BOOL	short [-1/0]	Boolean [True/False]	boolean [true/false]

Tabelle 3.6: Die COM-Basistypen [Box98]

Speziell für die ActiveX-Automation stellt COM eine gekapselte Union zur Verfügung, die VARIANT heißt. Dieser Datentyp stellt eine Teilmenge der von der IDL benutzten Datentypen bereit. Für jeden unterstützten Datentyp existiert eine Variablendeklaration. Welche Variable den Wert enthält, wird über ein zusätzliches Flag der Variantstruktur bestimmt. Tabelle 3.7 zeigt die zur Verfügung stehende Auswahl.

Zusätzlich können die beiden folgenden Flags mit den obigen Flags kombiniert werden. Damit kann angegeben werden, ob es sich bei dem beinhalteten Wert um eine Referenz oder um ein Feld von Werten des zugrundeliegenden Datentyps handelt (Quelle [Box98]).

VT_ARRAY Variant enthält ein Array von Werten
VT_BYREF Variant übergibt die Referenz des Wertes.

Für die Modellierung von Aggregations- und Assoziationsbeziehungen, die wiederum als Variablen eines Objekttypes modelliert werden, stellt die ActiveX Automation die Schnittstellenzeiger zur Verfügung. Damit ist der Zugriff zwischen den einzelnen Objekten gewährleistet.

Die Vererbung wird von der Automation nicht unterstützt. Das Fehlen der Vererbungsbeziehung stellt aber keine Einschränkung dar, da diese Beziehung hauptsächlich für die Implementierung

VT_EMPTY	leer
VT_NULL	Nullwert im SQL-Stil
VT_I2	short
VT_I4	long
VT_R4	float
VT_R8	double
VT_CY	CY (64-Bit Währungseiheit)
VT_DATE	DATE double
VT_BSTR	BSTR
VT_DISPATCH	IDispatch*
VT_ERROR	HRESULT
VT_BOOL	VARIANT_BOOL (true = -1, false = 0)
VT_VARIANT	VARIANT*
VT_UNKNOWN	IUnknown*
VT_DECIMAL	16-Byte Wert mit fester Dezimalstellenanzahl
VT_U11	Byte

Tabelle 3.7: ActiveX Datentypen (Quelle [Box98])

von wiederverwendbarem Code benutzt wird. In diesem Fall kann aber von einem fertigen Produkt ausgegangen werden, so daß die einzelnen vererbten Komponenten feste Bestandteile der übernommenen Objektstruktur sind.

Schwierigkeiten bereitet das objektorientierte Konzept, der Polymorphie. Zwar kann man über die Schnittstellenzeiger in einem betrachteten Objekt auf polymorphe Objekte verweisen, aber man findet keine Kennzeichnung in der Automation, um dieses Konzept zu erkennen. Da dieses Konzept auf einer Typbeschreibung eines gemeinsamen Oberobjektes für die polymorphen Objekten aufbaut, können im Integrationsmodul die Attribute des betrachteten Objektes angezeigt werden, die die polymorphen Objekte gemeinsam besitzen. Aus diesen Attributen muß eine Methodik ableitbar sein, um daraus verschiedene Auswahlsätze zu generieren, die die polymorphen Objekte mit der gleichen Typbeschreibung zusammenfassen.

Demnach stehen dem Entwickler ausreichende Möglichkeiten zur Verfügung, die verwendeten Datentypen in die Datentypwelt von COM abzubilden. Für die atomaren Datentypen stellt COM ausreichende Datentypen zur Verfügung. Für die Übergabe von Beziehungen zwischen den Objekten und Tupeltypen kann der Entwickler auf das Konzept der Schnittstellenzeiger zurückgreifen. Die Übernahme von LISTEN, MENGEN und ARRAYS beruht auf der Übernahme von ARRAYS, da zum Zeitpunkt der Übernahme die Anzahl der Elemente bekannt ist.

Letztendlich werden im Prototyp nur die atomaren Datentypen (einschließlich Schnittstellenzeiger für die Übernahme aggregierter oder assoziierter Strukturen) und Arrays verwendet. Die Tabelle 3.8 stellt die verwendeten Datentypen den Datentypen von COM gegenüber. Aus Aufwandsgründen konnte nicht die volle Palette der von COM benutzten Datentypen im Prototyp berücksichtigt werden. Die verwendeten Datentypen ergaben sich aus der Analyse der Schnittstellen-Beschreibung des Programmes AutoCAD R14, welches als Referenzbeispiel mit dem Integrationsmodul gekoppelt wurde.

Die persistente und die temporäre Implementierung der Datentypen unterscheiden sich im Integrationsmodul, da diese Daten mit unterschiedlichen Aufgaben konfrontiert werden. Zum einen werden die Daten in der übernommenen abstrakten Form für die Konsistenzkontrolle benötigt, zum anderen werden die importierten Daten dem Anwender für die weitere Bearbeitung zur Verfügung gestellt. Des weiteren ermöglicht diese Aufteilung eine einfache Handhabung der par-

ActiveX Datentypen	Datentypen im Integrationsmodul
VT_I2	Integer
VT_I4	Long
VT_R4	Float
VT_R8	Double
VT_BOOL	Boolean
VT_BSTR	String
VT_VARIANT	$\mathcal{K}_L$ und atomare Datetypen
VT_DISPATCH	$\mathcal{K}_T$, $\mathcal{K}_{Ref}$

Tabelle 3.8: verwendete ActiveX Datentypen im Integrationsmodul

allelen Bearbeitung durch mehrere Anwender. Die persistenten Daten können von den Anwendern nicht verändert werden, bilden aber den Ausgangspunkt für die weitere Bearbeitung. Für jeden Anwender kann ein neuer Datenraum für die temporären Daten erzeugt werden. Dadurch ist ein konfliktfreies Arbeiten weitestgehend möglich.

Zu den Aufgaben der Konsistenzprüfung gehört die Ermittlung der aus veränderten Objekten bestehenden Mengen, die sich aus einer Manipulation einer Planungsunterlage ergeben haben. Der Schwerpunkt liegt bei der Implementierung auf den Eigenschaften der Attribute und auf dem bestmöglichen Zugriff auf sie. Die Eigenschaften ergeben sich in diesem Zusammenhang aus den Aussagen, ob das Attribut schreibgeschützt (read-only) oder für die Identifizierung des Objektes in der Planungsunterlage benutzt wird.

Der Zugriff auf ein Attribut erfolgt über die Angabe der Applikation, das Modell und letztendlich durch die Angabe der gewünschten Instanz. Demnach wird von der Implementierung ein einheitliches Vorgehen für alle Attributzugriffe gefordert.

Zu den Besonderheiten der Implementierung im persistenten Datenraum gehören die Vergabe der eindeutigen Objektidentifikatoren, die im Integrationsmodul für die Konsistenzkontrolle benötigt werden, sowie die Festlegung, daß der Anwender des Integrationsmoduls die Werte der Attribute nicht verändern kann. Die Veränderung der Attributwerte kann nur durch eine Aktualisierung einer Planungsunterlage erfolgen.

Im Unterschied zur persistenten Implementierung der Daten erlaubt die temporäre Implementierung die Manipulation der Datenwerte. Die Implementierung muß für die einzelnen Datentypen polymorphe Operationen, beispielsweise für die Manipulation des Wertes, bereitstellen. Zweckmäßig ist die Bereitstellung der Datentypen mit den entsprechenden Operationen in separaten Klassen. Des weiteren empfiehlt es sich, den Zugriffspfad zu verändern. Da die Manipulation transaktionsorientiert ist, sollte der Zugriff auch bei einem Auswahlsatz beginnen.

Zu den Besonderheiten der temporären Implementierung gehört eine andere Behandlung der Objektidentitäten. Jedes Objekt bekommt im temporären Datenraum eine neue Identität. Die

persistenten Identitäten werden für die Definition von Beziehungen zwischen dem temporären Objekt und den persistenten Objekten benötigt. Da ein temporäres Objekt beliebig viele Beziehungen zu persistenten Objekten besitzen kann (mindestens eine), muß jedes temporäre Objekt ein zusätzliches Attribut besitzen, in dem die Menge der Objektidentifikatoren von den persistenten Objekten zusammengefaßt werden, die für die Erzeugung der aktuellen Instanz im temporären Datenraum verwendet wurden.

3.4 Objektidentitäten

Für die detaillierte Beschreibung von Veränderungen im Integrationsprozeß muß jedes Objekt in einem Modell eindeutig identifizierbar bleiben. Demnach müssen die Objektidentitäten zum persistenten Datenraum des Integrationsmodells gehören. Im Integrationsmodul reicht die bestehende Identifizierung der Objekte in den einzelnen Modellen nicht aus. Man kann nicht ausschließen, daß Objekte in verschiedenen Modellen gleiche Identitäten besitzen. Aus diesem Grund muß im Integrationsmodul der Identifizierungsmechanismus eines Objektes angepaßt werden.

Eine Identität besteht im Integrationsmodul aus mehreren Teilen. Der erste Teil ergibt sich aus einem Schlüssel, der jedes Modell im Integrationsmodul eineindeutig identifiziert. Der letzte Teil ergibt sich aus der Objektidentität des Objektes im speziellen Modell. Die Konkatenation beider Identitäten erlaubt eine eineindeutige Identifizierung aller Objekte.

Für die Arbeit mit den Identitäten ist es zweckmäßig, diese Identifizierung von Objekten um noch einen Identitätsschlüssel zu erweitern. Der zusätzliche Schlüssel bildet die einzelnen Objekttypen in einem Schlüsselwert ab. Damit enthält jedes Objekt in seiner Identifikation den konkreten Bezug zu seiner Typbeschreibung. Die endgültige Identität eines Objektes besteht somit aus drei Teilen:

- Modellidentität,
- Typidentität,
- Objektidentität.

Die Generierung der eindeutigen Datenwerte für die Modell- und Typidentifikation erfolgt im Integrationsmodul. Wann immer ein neues Modell in das Integrationsmodul eingefügt wird, wird ein neuer eindeutiger Wert für das Modell erzeugt. Ähnlich geschieht das mit der Typbeschreibung. Die Objektidentitäten werden in der entsprechenden Applikation gebildet und dürfen nicht im Integrationsmodul erzeugt werden. Würde das Integrationsmodul diese Identifikatoren erzeugen, so müßte dieser Wert in den Datenraum der Applikation eingefügt werden. Diese Forderung würde die Struktur des Datenraums der Applikation um ein Attribut erweitern, welches nicht zu den Modellierungsrichtlinien der Applikation gehört.

Da die Typbeschreibungen für die Objektidentifikation in den einzelnen Applikation nicht notwendigerweise den gleichen Aufbau besitzen, ist die Bildung eines Surrogatwertes zweckmäßig. Die Bildung des Surrogatwertes realisiert eine eineindeutige Abbildung, welche die verschiedenen Strukturbeschreibungen, die für die Identifizierung der Objekte in den Applikationen verwendet werden, in eine einheitliche Strukturbeschreibung im Integrationsmodul überführt. Diese Surrogatwerte sind im Integrationsmodul eineindeutig. Somit ist es möglich, ausgehend von einer

Identifizierung eines Objektes in einem Modell, den Surrogatwert des Objektes im Integrationsmodul zu ermitteln und auch umgedreht.

Die Identifizierung eines Objektes erfolgt nach folgendem Schema:

Surrogat(ModellIdentität.TypIdentität.ObjektIdentität).

Der Surrogatwert wird jedem Objekt in einem zusätzlichen Attribut bei der Erzeugung des Objektes im Integrationsmodul zugewiesen und persistent im Datenraum des Integrationsmoduls abgelegt. Dieser Wert wird verwendet, um Beziehungen zwischen den einzelnen Objekten zu modellieren.

3.5 Anforderungen an die Applikationen

Die Anwendung der COM und ActiveX-Automation Technologien bedingt, daß die Applikationen diese Technologien unterstützen müssen. Die ActiveX-Automation wird benutzt, um für ausgewählte Objekte die Objektbeschreibung sowie den Zustand des Objektes in das Integrationsmodul zu übernehmen. Mittels der COM-Technologie wird der Aufruf der hierzu benötigten Funktionalitäten realisiert, die von den Anwendungen bereitgestellt werden und im Integrationsmodul für die Aufgaben der Integration benötigt werden. Hierzu zählen Funktionalitäten für das Integrieren der Planungsunterlagen (Object Linking and Embedding) in das Integrationsmodul sowie spezielle Methoden, wie beispielsweise der Zugriff auf ein bestimmtes Objekt. Eine vollständige Beschreibung aller speziellen Methoden kann im Abschnitt 4.3 auf Seite 112 nachgelesen werden.

Für alle Objekte einer Anwendung, die am Datenaustausch beteiligt werden sollen, empfiehlt sich eine parametrische Beschreibung. Das bedeutet, daß alle Informationen in Form von veränderlichen Attributwerten angegeben werden. Die Veröffentlichung dieser Attribute mittels einer ActiveX-Automation Schnittstelle läßt sich dadurch auf eine geeignete Weise realisieren. Damit wird betont, daß die besten Ergebnisse mit den Objekten erzielt werden, welche nach objektorientierten und komponentenbasierten Konzepten modelliert sind. ActiveX-Automation Methoden können die Objekte besitzen, werden aber vom Integrationsmodul ignoriert (vergleiche mit Abschnitt 3 auf Seite 35).

Über die Behandlung von Beziehungen in der ActiveX-Automation Schnittstelle können folgende Regeln formuliert werden. Die Vererbung wird nicht in ActiveX unterstützt. Da die Vererbung eine Beziehung für die Wiederverwendung von Code darstellt, impliziert das Nichtvorhandensein keine Einschränkung für die Arbeiten im Integrationsmodul. Assoziationen dienen als Wege, auf denen Informationen von einem Objekt zum anderen Objekt transportiert werden [Booch94]. Diese Informationswege spielen für die Aufgaben der Integration eine untergeordnete Rolle. Aber genau wie Aggregationen können Assoziationen in der ActiveX-Automation Schnittstelle (siehe Abschnitt 3.3.2 auf Seite 62) angegeben werden.

Des weiteren müssen die Objekte, die an der Integration beteiligt werden sollen, über eine grafische Präsentation verfügen, damit der Anwender die Objekte in der Benutzeroberfläche auswählen kann. Für jedes Objekt muß eine Typbeschreibung in der Spezifikation der ActiveX-Automation existieren. Jede Typbeschreibung muß einen eindeutigen Namen besitzen, mit dem

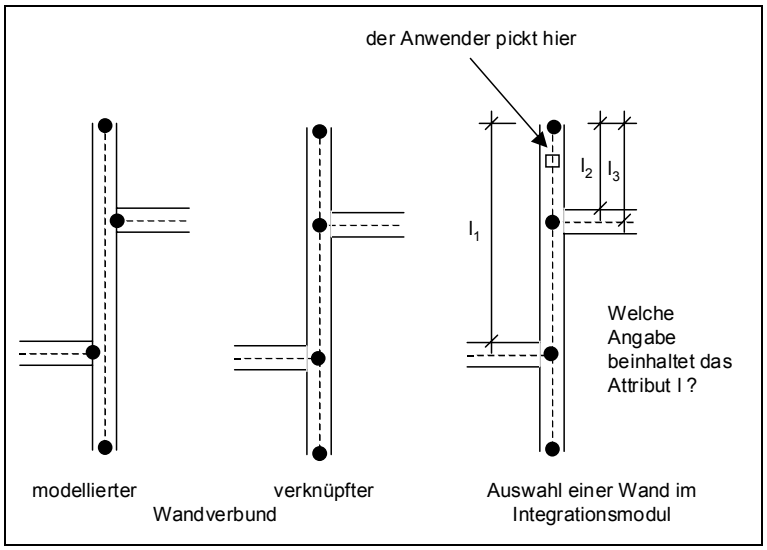


Abbildung 3.7: semantische Beschreibung des Inhaltes eines Wandattributes

eine Klassifizierung der Typen möglich ist. Die strukturelle Zusammensetzung der Typbeschreibung ergibt sich aus den elementaren Datentypen und aus der Anwendung von Typkonstruktionen. Für jedes Attribut der Typbeschreibung sollte ein Property-Eintrag in der ActiveX-Schnittstelle existieren. Diese Forderungen stellen keine Probleme dar, wenn die Applikation nach den objektorientierten Konzepten entwickelt ist.

Eine weitere Anforderung ergibt sich aus dem Nachrichtenkonzept. Jede Applikation muß die persistente Speicherung von Objektidentifikatoren unterstützen. Aus Performancegründen ist die Überwachung aller möglichen Attribute von allen Objekten nicht zweckmäßig (siehe Abschnitt 4.2.5 auf Seite 98).

Die nächste Forderung verlangt eine nachvollziehbare Abbildung zwischen den übernommenen Daten und deren visuellen Präsentation. Die Applikationen müssen beschreiben, welche Veränderungen sie am Datenraum in Abhängigkeit der aktivierten Aktion durchführen, wenn die Eingabedaten durch eine Berechnung verändert werden. Die Abbildung 3.7 zeigt ein Beispiel, indem nicht aus der grafischen Präsentation der Wert für die Länge einer Wand abgeleitet werden kann. In diesem Fall wird gefordert, daß die Anwendung den Wert angibt, der von den Anwendern eingegeben wurde ([Roth87]), oder eine Beschreibung der Wertangabe dem Anwender zur Verfügung stellt. In diesem Fall wäre beispielsweise die Länge der Mittellinie, also vom modellierten Knoten des Wandanfangs bis zum Knoten des Wandendes, zweckmäßig.

Jede Applikation muß die Generierung von Objekten durch die ActiveX-Automation erlauben. Damit wird indirekt gefordert, daß die Erzeugung eines Objektes auch ohne die Angabe eines einzigen Attributwertes durchgeführt werden kann. Das bedeutet, daß diese Objekte für die interne Datenverwaltung der Applikation keine Probleme darstellen.

Der Entwurf der ActiveX-Schnittstelle sollte auf die Modellierung von Containerklassen weitestgehend verzichten. Für den Aufbau dieser Klassen existiert derzeit noch kein Standard. Deshalb ist es unmöglich, ein einheitliches Vorgehen anzugeben, um auf die Elemente eines Containers

zuzugreifen. Das Integrationsmodul kann demnach solche Klassen nicht verarbeiten. Abschnitt 6.2.2 auf Seite 166 geht noch einmal detailliert auf diesen Aspekt ein.

Die nächste Forderung ergibt sich aus dem Blickwinkel, die Anwendung im Integrationsmodul während der Aktualisierung von Veränderungen optimal einsetzen zu können. Sie hat zum Gegenstand, daß gelöschte Objekte nicht aus dem Datenraum entfernt werden, sondern nur als gelöscht markiert werden. Damit ist es einem Anwender des Integrationsmoduls während der Aktualisierung möglich, auf gelöschte Objekte zuzugreifen und somit schneller die Veränderung in den Planungsunterlagen zu erfassen.

4 Spezifikation des Integrationsansatzes

4.1 Spezifikation des Integrationsmoduls - Überblick

Entsprechend dem Abschnitt 3.2 können drei Anwendungsfälle angegeben werden. Abbildung 4.1 zeigt die entsprechenden externen Schnittstellen des Integrationsmoduls sowie die drei erkannten Anwendungsfälle.

Basierend auf diesen Anwendungsfällen und den gewünschten Aktionen des Anwenders können die notwendigen Ereignisse spezifiziert werden. Demnach müssen folgende Kommandos in der Benutzeroberfäche des Integrationsmoduls dem Anwender zur Verfügung gestellt werden:

- für die Initialisierung von Applikationen und Modellen
  - Einfügen einer neuen Applikationsreferenz
  - Löschen einer Applikationsreferenz
  - Einfügen eines neuen Dokumentes in den Container
  - Entfernen eines Dokumentes aus dem Container
  - Anzeigen/Schließen der Präsentation eines ausgewählten und eingebetteten Dokumentes
- für die Übernahme von Objekten
  - Übernahme von Objekten aus einem Dokument in das Integrationsmodul
  - Übernahme von Objekten aus dem Integrationsmodul in ein Dokument
  - Generieren eines neuen Kommandos zur Manipulation übernommener Objekte
  - Ausführen eines Kommandos (Manipulation von Objekten eines Auswahlsatzes im Integrationsmodul)
  - Darstellen der Objekte eines Auswahlsatzes
- für die Wahrung der Konsistenz
  - Darstellung des Zustandes eines Dokumentes
  - Darstellung des Zustandes eines gewünschten Objektes
  - Aktualisieren eines vollständigen Dokumentes durch das Einbringen eines neuen Freigabestandes in das Integrationsmodul
  - Aktualisieren ausgewählter Objekte durch die im Integrationsmodul bereitgestellten Funktionalitäten.

Jede Ereignismethode implementiert einen separaten Prozeß, die miteinander über die persistenten oder dynamischen Datenstrukturen des Integationsmoduls in Verbindung stehen. Im folgenden wird für jede Ereignismethode der Inhalt des Prozesses angegeben.

Die Ereignismethoden des Anwendungsfalles 'Initialisierung von Applikationen und Modellen' besitzen folgende Spezifikation:

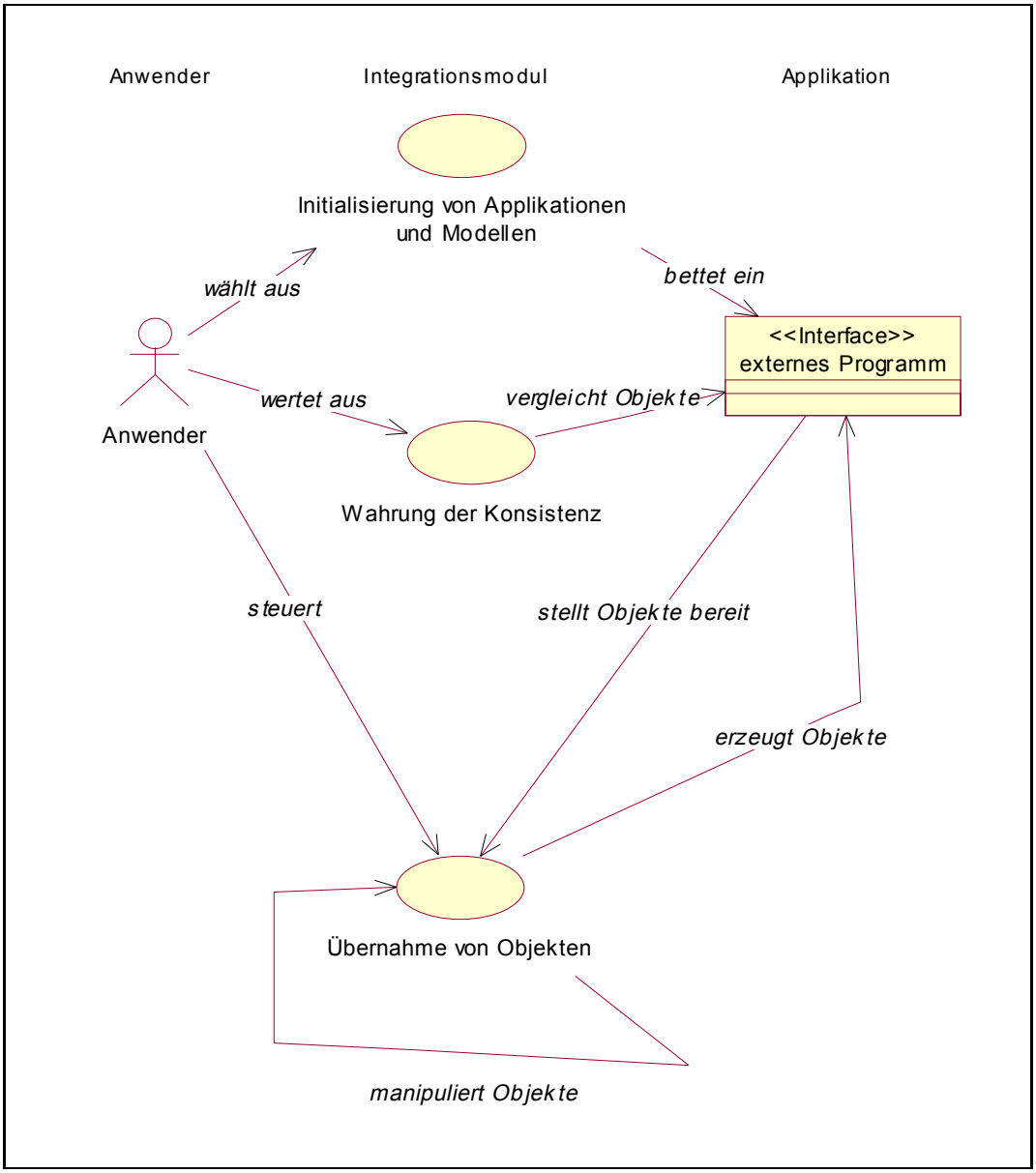


Abbildung 4.1: Anwendungsfälle des Integrationsmodul

Einfügen einer neuen Applikationsreferenz: Die Ereignismethode ermöglicht die Eingabe von Daten, die für die Referenzierung einer Applikation mittels COM benötigt werden. Das Ende des Prozesses erkennt der Anwender daran, daß die Viewkomponente des Integrationsmoduls die Applikation durch einen eindeutigen Namen in einem Treeview anzeigt.

Löschen einer Applikationsreferenz: Die Auswahl einer Applikation im Treeview der Viewkomponente des Integrationsmoduls und das Auswählen dieser Ereignismethode entfernt die entsprechende Applikationsreferenz aus dem persistenten Datenraum des Integrationsmoduls sowie alle zu dieser Applikation gehörenden Objekte. Zu den Objekten gehören alle Klasseninstanzen und alle eingebettete Dokumente mit ihren Objekten. Nach der Beendigung des Prozesses fehlt im Treeview der entsprechende Eintrag. Die Ereignismethode benutzt demnach die Implementierung des Ereignisses 'Entfernen eines Dokumentes'.

Einfügen eines neuen Dokumentes: Das Einfügen eines neuen Dokumentes setzt die Auswahl einer Applikation voraus, um die Verbindung des Dokumentes zu dem entsprechenden Server aufbauen zu können. Mittels der existierenden Schnittstellen von OLE wird das Dokument persistent in den Datenraum des Integrationsmoduls eingebettet sowie die notwendigen Abhängigkeiten zur Applikation generiert. Durch einen eindeutigen Namen, der vom Anwender angegeben werden muß, läßt sich jedes Dokument in der Viewkomponente des Integrationsmoduls identifizieren. Das Einfügen eines neuen Dokumentes erfordert geeignete Maßnahmen für die Sicherung der Konsistenz. Aus diesem Grund erzeugt das Integrationsmodul eine globale Nachricht an alle bereits enthaltenen Dokumente, um die Eigentümer der bereits enthaltenen Dokumente über die aufgetretene Veränderung zu informieren. Dieser Prozeß unterstützt die Angabe eines leeren, noch keine Objekte enthaltenden Dokumentes sowie die Angabe eines im Dateisystem bestehenden Dokumentes.

Entfernen eines Dokumentes: Das Markieren eines Modellnamens im Integrationsmodul sowie die anschließende Auswahl dieser Ereignismethode hat zur Folge, daß das entsprechende Dokument mit den enthaltenen Objekten aus dem Integrationsmodul entfernt wird. Dieser Prozeß aktualisiert des weiteren die im Integrationsmodul enthaltenen Beziehungen, d.h. die Referenzen zwischen der Applikation und dem Dokument werden entfernt sowie alle Objekte des Dokuments werden im Integrationsmodul als gelöscht markiert und für die Weiterbearbeitung dem Prozeß 'Wahrung der Konsistenz' übergeben.

Anzeigen der Präsentation des Dokumentes: Gegenstand des Prozesses dieser Ereignismethode ist es, ein ausgewähltes Dokument in der fachspezifischen Notation darzustellen. Die zugrundeliegende Implementierung des Prozesses wertet hierzu die in der Ereignismethode 'Einfügen eines Dokumentes' angegebene Referenz aus, startet die entsprechende Applikation und stellt sicher, daß das eingebettete Dokument in der Applikation geladen wird. Hierfür stellt OLE die notwendigen Funktionalitäten zur Verfügung.

Schließen der Präsentation des Dokumentes: Diese Ereignismethode beendet die Bearbeitung des Modelles in der Applikation. Alle dynamisch erzeugten Verweise werden freigegeben.

Für die Ereignismethoden des Anwendungsfalls 'Initialisierung von Applikationen und Modellen' werden hauptsächlich Methoden für die Kommunikation eingesetzt, die zum Standardumfang von OLE gehören.

Die Spezifikation der Methoden des Anwendungsfalls 'Übernahme von Objekten' lautet:

Auswählen von Objekten: Die Ereignismethode hat zum Ziel, ausgewählte Objekte eines angegebenen Dokumentes in das Integrationsmodul zu importieren. Hierbei müssen die Objekttypen der ausgewählten Objekte sowie die Objekte selbst persistent im Integrationsmodul nachgebaut werden. Die einzelnen Objekttypen sowie die einzelnen Objekte werden separat in der Viewkomponente des Integrationsmoduls eingefügt, wobei die Klassen der zugrundeliegenden Applikation und die Objekte dem entsprechenden Modell zugeordnet werden.

Des weiteren erzeugt der Prozeß für jeden übernommenen Objekttyp einen neuen Auswahlsatz im dynamischen Datenraum des Integrationsmoduls und fügt die Objekte in den Auswahlsatz ein, die diesen Objekttyp verwenden.

Für das Auswählen der Objekte in einem angegebenen Dokument muß das Dokument in der fachspezifischen Notation dem Anwender zur Verfügung stehen. Diese Ereignismethode benutzt demnach den Prozeß 'Anzeigen der Präsentation des Dokumentes'.

Eine weitere Anforderung an den Prozeß ergibt sich aus der Möglichkeit, wiederholt Objekte auszuwählen, die bereits im Integrationsmodul persistent vorliegen. In diesem Fall dürfen weder der Objekttyp noch das Objekt selbst noch einmal persistent erzeugt werden.

Übernahme von Objekten: Diese Ereignismethode erzeugt neue Objektinstanzen in einem angegebenen Dokument. Hierbei muß ein Auswahlsatz mit den zu übernehmenden Objekten im Integrationsmodul angegeben werden. Für jedes Objekt des Auswahlsatzes wird eine neue Instanz im angegebenen Dokument erzeugt und die Attributwerte des Quellobjektes werden dem erzeugten Objekt zugewiesen. Im Anschluß daran werden die erzeugten Instanzen in den persistenten Datenraum des Integrationsmoduls übernommen und die notwendigen Abhängigkeiten zwischen den Objekten im Auswahlsatz und den erzeugten Instanzen persistent im Integrationsmodul modelliert. Voraussetzung hierfür sind die Angabe des Zieldokuments, die Angabe eines entsprechenden Objekttyps des Zieldokuments und die Definition von Abbildungsvorschriften zwischen dem Objektyp der Objekte im Auswahlsatz und dem angegebenen Objekttyp im Zieldokument.

Generiere ein neues Kommando: Die Ereignismethode erlaubt die Eingabe eines neuen Kommandos. Die einzelnen Kommandos werden im Integrationsmodul unter einem eindeutigen Namen abgelegt. Jedes Kommando operiert entweder auf einem Auswahlsatz oder auf einem bestehenden Kommando, welches durch den entsprechenden Namen angegeben wird.

Ausführen eines Kommandos: Das Aktivieren dieses Ereignisses erzeugt einen Auswahlsatz im dynamischen Datenraum des Integrationsmoduls entsprechend dem gewünschten Kommandoinhalt. Der Name des Auswahlsatz entspricht dem Namen des zugrundeliegenden Kommandos.

Darstellen der Objekte eines Auswahlsatzes: Dieser Prozeß erzeugt eine alternative Darstellung der Objekte eines angegebenen Auswahlsatzes in einer Tabelle.

Die Implementierung der einzelnen Ereignismethoden für den Anwendungsfall 'Übernahme von Objekten' setzt sich aus einzelnen Teilprozessen zusammen. Abbildung 4.2 benennt die einzelnen Teilprozesse und beschreibt die Beziehungen zwischen diesen.

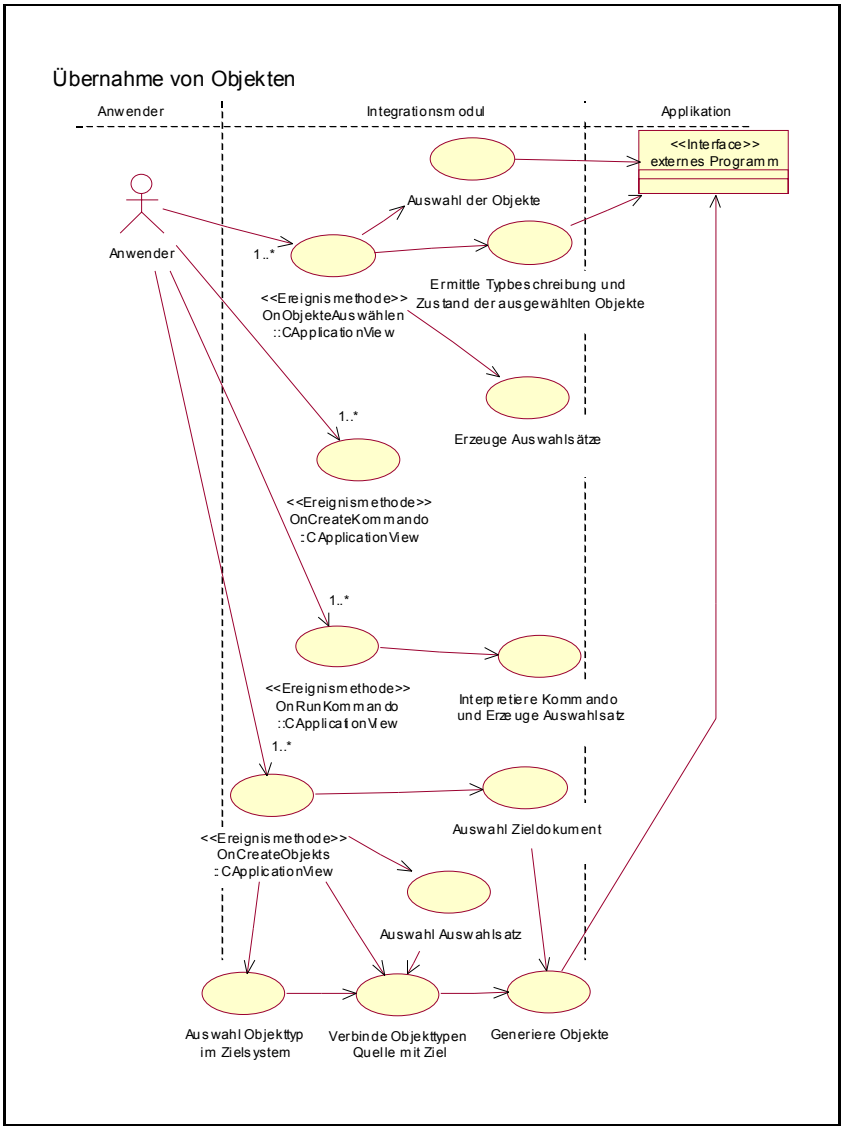


Abbildung 4.2: Prozeßbeschreibung: Übernahme von Objekten

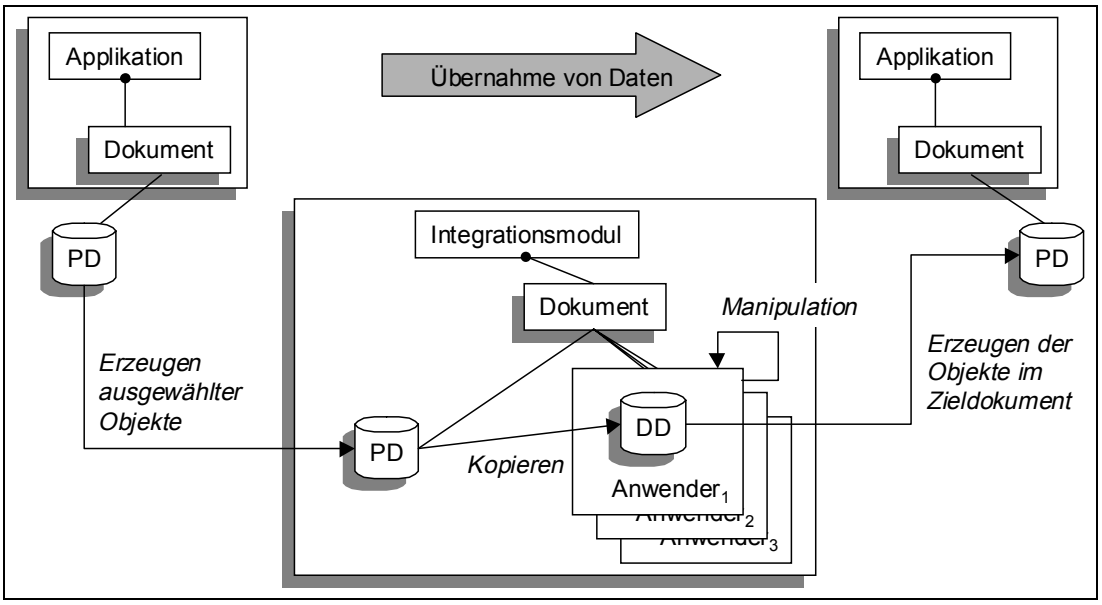


Abbildung 4.3: Informationsfluß des Anwendungsfalles 'Übernahme von Objekten'

Für die Realisierung der ersten beiden Ereignismethoden muß das Integrationsmodul mit der externen Applikation kommunizieren. Die restlichen Ereignismethoden operieren mit den im Integrationsmodul angelegten Instanzen und benötigen demnach keine Kommunikation mit einer Applikation.

Demnach müssen für die Teilprozesse 'Auswahl der Objekte', 'Ermittle Typbeschreibung und Objektzustand' und 'Generiere Objekte' neue Kommunikationsmöglichkeiten entwickelt werden, damit im Integrationsmodul die notwendigen Ereignisse implementieren werden können. Die Abschnitte 4.2.1, 4.2.2 und 4.2.4 gehen gezielt auf den Entwurf der neuen Kommunikationsschnittstelle ein.

Die Grammatik der Kommandosprache wird im Abschnitt 4.2.3 ab Seite 83 vorgestellt.

Die einzelnen Aktionen simulieren eine Prozeßkette, wobei jeder Prozeß bestimmte Daten von dem vorangegangenen Prozeß erhält und daraus neue Daten erzeugt. Die Prozeßkette (siehe Abbildung 4.3) beginnt mit dem Auswählen der Objekte in einem Quelldokument und endet mit dem Erzeugen der Instanzen in einem Zieldokument. Zwischen diesen beiden Prozessen können die übernommenen Objekte durch die Kommandosprache manipuliert werden.

Nach der Auswahl der Objekte werden entsprechende Kopien der ausgewählten Objekte im persistenten Dateraum (PD) des Integrationsmoduls angelegt. Diese persistenten Objekte werden für die 'Wahrung der Konsistenz' verwendet und können nicht von den einzelnen Anwendern verändert werden. Die Verwaltung der persistenten Objekte obliegt ausschließlich dem Integrationsmodul. Das Integrationsmodul ermöglicht die Veränderungen an den persistenten Daten durch entsprechende Aktualisierungsmethoden (Wahrung der Konsistenz). Zusätzlich erzeugt dieser Prozeß weitere Objektkopien, die im dynamischen Datenraum (DD) des Integrationsmoduls abgelegt werden, die von den einzelnen Anwendern im folgenden Prozeß beliebig bearbeitet werden können.

Die Bearbeitung der Objekte im dynamischen Datenraum muß die parallele Arbeitweise der Anwender berücksichtigen. Damit sich die Anwender, die zur gleichen Zeit mit den gleichen Objekten im Integrationsmodul arbeiten, nicht behindern, wird für jeden Anwender ein separater dynamischer Datenraum erzeugt. Jeder Datenraum beinhaltet nur die Objekte, die der Anwender entweder durch das Auswählen oder durch das Anwenden eines Kommandos auf die in seinem dynamischen Datenraum befindlichen Objekte erzeugt hat.

Der letzte Prozeß übernimmt eine Menge von typgleichen Objekten in das Zieldokument. Er erstellt demnach aus den im dynamischen Datenraum befindlichen Objekten persistente Objektkopien im Zieldokument.

Diese einzelnen Prozesse sind voneinander unabhängig. Die Verbindung erfolgt ausschließlich über die erzeugten Objekte in den jeweiligen Prozessen. Der Anwender kann demnach beliebig oft die einzelnen Prozesse aktivieren in der Reihenfolge, in der er die erzeugten Objekte für die folgenden Prozesse benötigt.

Die Spezifikation der Methoden des Anwendungsfalles 'Wahrung der Konsistenz' lauten:

Darstellung des Zustandes eines Dokumentes: Das Ereignis erzeugt einen View, der den Zustand eines Dokumentes in Abhängigkeit der von anderen im Integrationsmodul eingebetteten Dokumente darstellt. Inhalt des Views sind Informationen über den Zustand des Dokumentes, die einzelnen globalen Nachrichten sowie die Objekte, die einer Aktualisierung bedürfen.

Darstellung der Abhängigkeiten eines gewünschten Objektes: Das Aktivieren dieses Ereignisses erzeugt einen View, der ausgehend von einem bestimmten Objekt alle abhängigen Objekte und deren Zustände angibt. Zusätzlich werden noch die Art der Veränderung und der Status der Bearbeitung bezüglich zweier Objekte angegeben. Die Darstellung der Objekte kann mit Filtern eingeschränkt werden, die die Angabe von bestimmten Objekten mit ausgewählten Eigenschaften und Beziehungen beachten.

Aktualisieren eines Dokumentes: Diese Methode des Integrationsmoduls gehört zu den wenigen Ereignismethoden, die durch die externe Applikation aktiviert werden und die zum Standardumfang von OLE gehören. Gegenstand der Methode ist es, einen neuen Freigabestand eines Dokumentes in das Integrationsmodul einzubetten. Im Anschluß daran erfolgt die Bestimmung des Konsistenzzustandes von allen im Integrationsmodul enthaltenen Dokumenten.

Aktualisieren von Objekten: Diese Ereignismethode aktualisiert bestehende Objektinstanzen in einem angegebenen Dokument. Hierbei muß ein Auswahlsatz des Integrationsmoduls angegeben werden. Für jedes Objekt des Auswahlsatzes wird ein Zugriff auf die zugehörige Instanz im angegebenen Dokument erzeugt und die Attributwerte des Quellobjektes werden dem erzeugten Objekt zugewiesen.

Die ersten beiden Ereignismethoden des Anwendungsfalles 'Wahrung der Konsistenz' benötigen keine Kommunikation mit externen Applikationen. Diese Ereignismethoden werten die im Integrationsmodul angelegten Daten aus und stellen den Anwendern die gewünschten Übersichten zusammen.

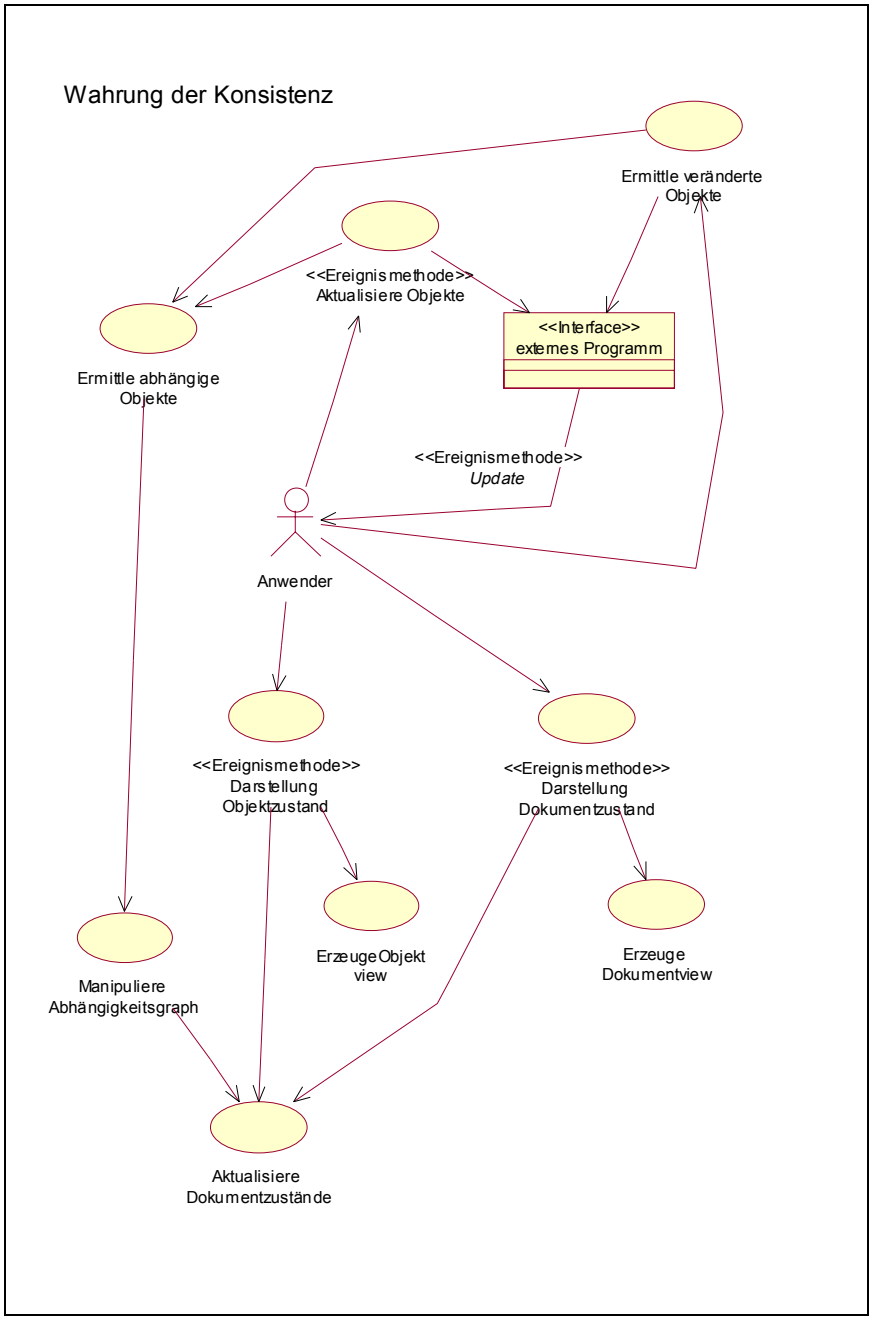


Abbildung 4.4: Prozeßbeschreibung: Wahrung der Konsistenz

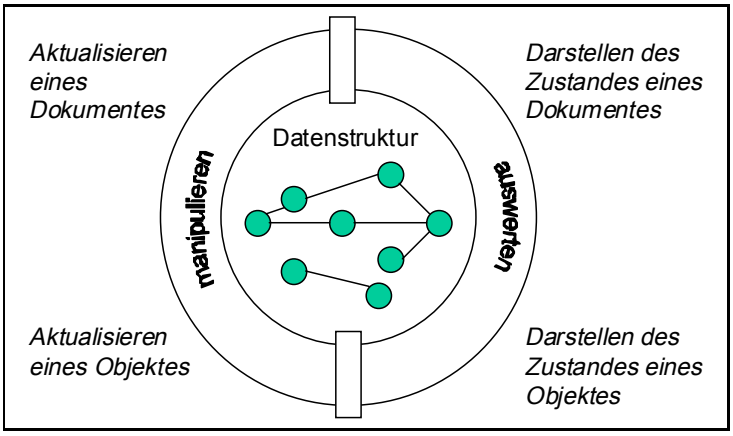


Abbildung 4.5: Informationsfluß des Anwendungsfalles 'Wahrung der Konsistenz'

Die Realisierung der Ereignismethoden setzt sich aus einzelnen Teilprozessen zusammen. Abbildung 4.4 benennt diese und stellt deren Abhängigkeiten dar. Der Abschnitt 4.2.5 ab Seite 96 geht detailliert auf die einzelnen Teilprozesse ein.

Im Gegensatz zu den Aktionen des Anwendungsfalles 'Übernahme von Objekten' operieren diese Aktionen auf einer gemeinsamen Datenstruktur, die entweder manipuliert oder ausgewertet wird. Der Datentyp dieser Datenstruktur basiert auf einem gerichteten Graphen, bestehend aus Knoten und Kanten. Die Knoten stellen die Objekte im persistenten Datenraum des Integrationsmoduls dar, die entweder verändert wurden oder die auf Grund einer Veränderung eines Objektes inkonsistent sind. Die Kanten setzen zwei Objekte in Beziehung, die die gleichen Informationsinhalte redundant darstellen. Die Richtung der Kante legt fest, welches Objekt die Information festlegt (Startknoten der Kante) und welches Objekt die Informationen verwendet (Endknoten der Kante).

Für die Manipulation des Graphen ist die Behandlung von Zyklen ausschlaggebend. Für das Auswerten des Graphen spielt die zeitliche Reihenfolge der Aktionen eine dominante Rolle.

Das Auftreten von Zyklen in Graphen ist auf Grund der eindeutigen Regelungen der Vorgabeberechtigung nicht möglich. Trotzdem ist es angebracht, entsprechende Algorithmen für das Überprüfen des Graphen vorzusehen, sobald neue Kanten in den Graphen eingefügt werden.

Die Gewährleistung, daß alle Aktionen immer auf einen aktuellen Zustand des Graphen zugreifen können, wird über den Manager des Graphen sichergestellt. Der Manager erlaubt nur einem Prozeß zu einem bestimmten Zeitpunkt, den Graphen zu manipulieren. Lesende Prozesse werten die Datenstruktur nur aus und bekommen hierfür eine aktuelle und unveränderbare Kopie des Graphen.

Abbildung 4.5 verdeutlicht noch einmal den Informationsfluß.

4.2 Spezifikation der Teilprozesse

4.2.1 Auswahl der Objekte

Inhalt des Prozesses ist es, dem Anwender die Auswahl der Objekte in einem Modell zu ermöglichen. Der Prozeß muß, egal welche Applikation dem Modell zugrundeliegt, die spezielle, optimierte Variante für die Identifizierung der Objekte dem Anwender zur Verfügung stellen. Resultat des Prozesses ist der Zugriff auf eine Menge von Objekten. Diese Objekte werden danach in den persistenten Datenraum des Integrationsmoduls übernommen, indem sie physisch nachgebaut werden.

Je nachdem, wie die einzelnen Objekte eines Dokumentes in der Benutzeroberfläche dargestellt werden, existieren unterschiedliche Verfahren für die Identifizierung.

Befinden sich die Objekte in einem grafischen Editor, beispielsweise CAD, kann die Identifizierung der Objekte aus einer Kombination von verschiedenen Möglichkeiten erfolgen, wie durch die sequentielle Auswahl einzelner Objekte mittels der Eingabe eines zu dem Objekt gehörenden Punktes oder durch die Auswahl aller Objekte, die sich beispielsweise innerhalb eines durch die Eingabe zweier Punkte aufgezogenen achsenparallelen Rechteckes befinden.

In einem relationalen Editor, z.B. einem Frontend einer Datenbank, erfolgt die Identifizierung der Objekte mittels der Zugehörigkeit zu einer Relation. Demnach erfolgt beispielsweise in relationalen Datenbanken die Auswahl der Objekte durch die Angabe einer Tabelle oder durch die Angabe einer Select-Abfrage.

Um die verschiedensten Möglichkeiten der Identifizierung unterstützen zu können, ist es zweckmäßig, die Implementierung der Methode der entsprechenden Applikation zu überlassen. Der Prozeß bekommt einen Verweis auf die entsprechende Methodenimplementierung. Die Voraussetzung hierfür ist, daß die Signatur der Methode bekannt und fest vorgegeben ist. Die Signatur in C++ lautet:

```
int SelectOnScreen( ISelectedObjects **pobjects);
```

Ein negativer Rückgabewert beschreibt einen Fehlercode. In allen anderen Fällen beinhaltet der Rückgabewert die Anzahl der Elemente in der Liste, die über den Parameter der Methode übernommen wurden. Das Interface ISelectedObjects erlaubt einen sequentiellen Zugriff auf die einzelnen Elemente der Liste.

Der Anwender muß für die korrekte Abarbeitung des Ereignisses zwei Informationen bereitstellen. Zum einem ist es die Auswahl des Modells und zum anderen die Vergabe der Namen für den Auswahlsätze, unter denen später die ausgewählten Objekte angesprochen werden können.

Nachdem diese Informationen bereitgestellt sind, übernimmt dieser Prozeß die Kontrolle. Er aktiviert das entsprechende Modell, d.h. die zum Modell gehörende Applikation wird gestartet und das Modell dargestellt. Danach wird die Referenz zur Methodenimplementierung von *SelectOnScreen* aufgebaut und die Methode gestartet.

Im Anschluß daran befindet sich dieser Teilprozeß im Wartezustand. Währenddessen erfolgt im Server das Auswählen der gewünschten Objekte sowie die Zusammenfassung der identifizierenden

Schnittstellenzeiger der ausgewählten Objekte zu einer Liste. Die Liste wird über die Schnittstelle ISelectedObjects veröffentlicht.

Das Interface ISelectedObjects stellt dem Integrationsmodul neben den Standardfunktionalitäten eines Interface' noch folgende Funktionalitäten des Servers zur Verfügung:

```
HRESULT ISelectedObjects::IsNotEmpty();
HRESULT ISelectedObjects::Head(IDispatch **pDisp);
HRESULT ISelectedObjects::getClassName(IDispatch *pDisp, BSTR *classname);
HRESULT ISelectedObjects::IsIdentKey(BSTR classname, BSTR AttributeName);.
```

Die ersten beiden dienen dem schrittweisen Abarbeiten der Liste. Mit der ersten Methode wird getestet, ob die zugrundeliegende Liste noch Elemente beinhaltet. Ein Wert größer Null gibt die Anzahl der noch vorhandenen Listenelemente zurück. Beinhaltet die Liste keine Elemente, so liefert die Methode den Wert Null zurück. Mit der zweiten Methode kann das jeweils erste Listenelement erfragt werden. Die Methode liefert einen Zeiger auf das OLE-Automation Interface des Objektes zurück und entfernt das entsprechende Listenelement aus der Aufzählung.

Die vorletzte Methode wird benötigt, um den Namen der Strukturbeschreibung des Objektes in der TypeLibrary zu ermitteln. Die letzte Methode ermöglicht das Erkennen der Attribute, die für die Identifizierung des Objektes im Dokument verwendet werden. Hierzu wird der Name der Strukturbeschreibung und ein weiterer Name angegeben, der ein Attribut der entsprechenden Strukturbeschreibung bezeichnet. Liefert die Methode einen Wert größer Null zurück, so wird das Attribut in der angegebenen Klassenbeschreibung für die Identifizierung des Objektes im Dokument verwendet.

4.2.2 Erzeugen der Objekte im Integrationsmodul

Gegenstand des Teilprozesses ist es, die ausgewählten Objekte persistent in den Datenraum des Integrationsmoduls zu übernehmen und die erzeugten Objekte sowie deren Typen grafisch darzustellen. Der Prozeß bekommt den Zeiger *pobjects* übergeben, welcher auf einzelne ausgewählte Objekte in einem Dokument verweist, und erstellt eine Aufzählung von Objektidentifikatoren, die im Integrationsmodul erzeugt worden sind und die die ausgewählten Objekte im Integrationsmodul eindeutig identifizieren. Diese Liste mit Objektidentifikatoren wird dem folgenden Prozeß unter dem Namen *pObjectIds* zur Verfügung gestellt.

Die Visualisierung der erzeugten Objekte und deren Typen basiert auf der Erweiterung des bei der Übernahme der Applikationen und Modelle erzeugten Baumes. Da gewöhnlich eine Applikation ein unveränderliches Typsystem unterstützt, ist es zweckmäßig, die erzeugten Objekttypen direkt unter den Root-Knoten des entsprechenden Eintrages der zugrundeliegenden Applikation einzufügen. Jeder Objekttyp erzeugt einen neuen Teilbaum, der die Struktur des Typs wiedergibt. Für die Darstellung der Objekte verhält es sich ähnlich. Für jedes Objekt wird ein separater Teilbaum erzeugt, dessen Rootknoten die Identifizierung des Objektes darstellt. Diese Teilbäume werden getrennt nach der Zugehörigkeit zu einem Objekttyp unterhalb des zugrundeliegenden Dokumentes eingefügt (siehe Abbildung 5.4 auf Seite 121).

Die Erzeugung der einzelnen Bäume beruht auf der Regel, daß für jeden zusammengesetzten Datentyp ein neuer Knoten in den Baum eingefügt wird, dessen Blätter die einzelnen Komponenten darstellen.

Die Erzeugung der Objekte und Objekttypen im Integrationsmodul beruht auf einem Metakonzept, wobei für jeden Datentyp der OLE-Automation ein Repräsentant im Metamodell vorliegt. Tabelle 3.7 auf Seite 62 stellt die vom Integrationsmodul unterstützten Metaklassen dar.

Der grundlegende Algorithmus, der in diesen Teilprozeß implementiert wurde, besitzt folgenden Aufbau:

1. WHILE pObjects->IsEmpty() > 0
   - pObjects->Head(&pDisp)
   - pObjects->getClassName(pDisp, ClassName);
   - IF Existiert die Klasse mit den Namen ClassName
     - Erzeuge einen Zugriff auf die persistente Klassenbeschreibung im Integrationsmodul
   - ELSE
     - Erzeuge eine neue persistente Klassenbeschreibung im Integrationsmodul
     - Stelle die erzeugte Klassenbeschreibung in der Benutzeroberfläche dar
   - ENDIF
   - Erzeuge eine neue Instanz basierend auf der Klassenbeschreibung
   - Übernehme die Attribute
   - IF existiert die Instanz im Integrationsmodul
     - Ermittle die Identifikation des Objektes im Integrationsmodul
   - ELSE
     - Erzeuge eine neue Objektidentifikation
     - Speichere die erzeugte Instanz unter der neu generierten Objektidentifikation
     - Stelle das erzeugte Objekt in der Benutzeroberfläche dar
   - ENDIF
   - Trage die Objektidentifikation in die Liste *pObjectIds* ein
2. EWHILE.

Die Anweisung 'Erzeuge eine neue Klassenbeschreibung' setzt die im Integrationsmodul vorhandenen Metaklassen wie Bausteine zu einem neuen Objekttyp zusammen. Hierzu wird der Verweis zu den entsprechenden Einträgen in der Typelibrary verwendet, der laut OLE-Automation Spezifikation für jedes Dispatch-Interface standardmäßig vorhanden ist. Alle Einträge in der Typelibrary zu einem bestimmten Dispatch-Interface lassen sich sequentiell ermitteln.

In einem Iterationsschritt wird somit die Dokumentation für ein Element des Interfaces analysiert. Die Dokumentation beinhaltet folgende Informationen:

Name des Elementes: ASCII-Repräsentation des Elementes.

Art des Elementes: Beantwortet die Frage, ob das Element als Methode, als Attribut oder als Variable[1] verwendet wird.

Art des Zugriffes: Handelt es sich bei dem Element um ein Attribut, differenziert dieser Eintrag die Verwendung des Elementes entweder zum Lesen oder zum Schreiben.

Datentyp des Attributes: Tabelle 3.7 auf Seite 62 beinhaltet die möglichen Einträge dieses Segmentes.

Das prinzipielle Vorgehen stützt sich auf einen Algorithmus, der als erstes eine neue Klasseninstanz erzeugt und den Namen der Klassenbeschreibung dieser Instanz zuweist. Im Anschluß daran wird die Dokumentation des dem Objekt zugrundeliegenden Dispatch-Interfaces geöffnet. Anschließend werden nacheinander alle Elemente abgearbeitet, die von der Art her Attribute sind. Variablen und Methoden werden derzeit im Prototyp nicht unterstützt.

Für jedes Attribut wird der Name, der Datentyp und die Art des Zugriffes ausgelesen. Entsprechend des Datentyps wird eine neue Instanz erzeugt, basierend auf der im Integrationsmodul existierenden atomaren Datentyp-Metaklasse. Dieser Instanz wird der ermittelte Name zugewiesen. Existiert dieses Attribut bereits in der Klasseninstanz, so wird die Art des Zugriffes beider Attributinstanzen kombiniert.

Der letzte Schritt umfaßt die Markierung der Attribute, die für die Identifikation des Objektes im Dokument des Servers benutzt werden. Hierzu wird die neu erzeugte Klasseninstanz verwendet. Für jedes Attribut der Instanz wird der Wahrheitswert ermittelt, ob das Attribut für die Identifizierung verwendet wird oder nicht (siehe Methode IsIdentKey(..)).

Bei der Erzeugung einer neuen Klassenbeschreibung werden keine Beziehungen beachtet. Sollen referenzierte Objekte übernommen werden, so muß ausgehend von einer existierenden Klassenbeschreibung das Attribut weiter bearbeitet werden, welches die Beziehung zu dem gewünschten Objekt modelliert. Üblicherweise ist der Datentyp des Attributes ein Dispatch-Zeiger.

Der Grund für diesen zusätzlichen Bearbeitungschritt liegt darin, daß alle Objekte eine Referenz zu einem Containerobjekt beinhalten. Eine automatische Auswertung aller Beziehungen würde zur Übernahme des Containerobjektes führen. Die Übernahme des Containerobjektes wiederum führt zur automatischen Übernahme des Objektes, welches wiederum das Containerobjekt aggregiert. Das Endergebnis einer automatischen Übernahme der Beziehungen würde letztendlich zur vollständigen Übernahme aller Objekte des aktuellen Dokumentes führen, was nicht gewünscht ist.

Beziehungen nicht auszuwerten, wäre ein Verlust von modellierten Informationen. Diese Informationen im Integrationsmodul nachzubauen, ist nur schwer möglich. Eine Kompromißlösung stellt der Ansatz dar, den Anwender selbst entscheiden zu lassen, welche Beziehungen im Integrationsmodul aufgebaut werden sollen. Durch die Auswahl eines Attributes in der visuellen Darstellung einer importierten Klassenbeschreibung kann eine Aktion gestartet werden, die für alle importierten Objekte eines Modells die entsprechenden referenzierten Objekte importiert und dabei die entsprechenden Verweise im Integrationsmodul anlegt.

[1] Methoden simulieren das Verhalten des Objektes. Attribute und Variablen definieren in ihrer Gesamtheit den Zustand des Objektes. Der Unterschied zwischen einem Attribut oder einer Variablen liegt darin, daß der Zugriff auf einen Attributwert indirekt mittels Attributmethoden erfolgt währenddessen bei einer Variable der Zugriff direkt erfolgt.

Der Algorithmus für die Übernahme referenzierter Objekte ist ähnlich. Die entsprechende Instanz der Metaklasse $\mathcal{K}_{Ref}$, die die Schnittstellenzeiger im Integrationsmodul implementiert, beinhaltet deswegen zwei Klassenvariablen. Die erste Klassenvariable gibt Auskunft darüber, ob die referenzierten Objekte eines bestimmten Dokumentes übertragen werden sollen. Das zweite Attribut beinhaltet die Identifikation des referenzierten Objektes in der entsprechenden Objektinstanz, falls die Klassenbeschreibung der Objektinstanz die Übernahme des referenzierten Objektes beinhaltet.

Die Erzeugung der Objektinstanz basiert auf der neu angelegten Klasseninstanz. Jede Metaklasse, die für die Erzeugung einer Klasseninstanz verwendet werden kann, beinhaltet eine Methode *Copy* und eine Member-Variable *m_ Wert*. Der Aufruf der Methode *Copy* von der Klasseninstanz führt zum rekursiven Aufruf der Methode *Copy* von allen in der Klasseninstanz aggregierten Metaklassen und somit zur Erzeugung einer typgleichen Klasseninstanz, die als Objektinstanz bezeichnet wird.

Als nächstes muß der Zustand des Objektes im Dokument auf die Objektinstanz im Integrationsmodul übertragen werden. Hierfür wird sukzessiv für jedes Attribut die entsprechende Attributmethode des Dispatch-Interfaces aufgerufen, und der zurückgegebene Wert der entsprechenden Member-Variable *m_ Wert* zugewiesen. Eine Besonderheit hierbei ist die Erkennung einer Feldvariablen bei einem VARIANT-Typ. Die entsprechende Implementierung der Metaklasse, die einen VARIANT-Typ nachbaut, berücksichtigt diesen Aspekt durch eine Feldvariable, deren Größe dynamisch verändert werden kann. Eine weitere Besonderheit ist die Übernahme von referenzierten Instanzen. In diesem Fall wird das Objekt übernommen und dessen Identifikation dem entsprechenden Attribut des referenzierenden Objektes zugewiesen.

Die persistenten Datenstrukturen verfolgen das Ziel, den Entwurf der beteiligten Metaklassen auf das Typsystem der ActiveX Automation anzupassen. Damit wird gewährleistet, daß diese Datenstrukturen optimal für den Anwendungsfall 'Wahrung der Konsistenz' sind.

Die Aufgabe des Prozesses 'Übernahme der Objekte in den temporären Datenraum des Integrationsmoduls' ist es, die umständliche Datenstruktur des persistenten Metaklassen- und Instanzenkonzeptes in anwenderfreundliche Strukturen zu übertragen. Dieser Prozeß wird wegen der organisatorischen Gegebenheiten notwendig, die in der parallelen Arbeitsweise der spezialisierten Anwender begründet liegt, und auf Grund der verschiedenen, von den einzelnen Anwendern abhängigen Ziele, die mit der Übernahme von Daten erreicht werden sollen.

Die Optimierung des Entwurfes der temporären Datenstrukturen verfolgt das Ziel, eine gute Voraussetzung für die Bearbeitung der Objekte und deren Typen zu ermöglichen. Attribute vom Typ Schnittstellenzeiger werden in den Datentyp CIMO_REFERENCE umgewandelt, der spezielle Operationen für die Bearbeitung von Beziehungen ermöglicht. Attribute mit dem Typ VARIANT werden auf den zugrundeliegenden Typ der überladenen varianten Struktur gesetzt, da die entsprechende Typinformation erst mit der Übernahme eines Objektes ermittelt werden kann.

Den Ausgangspunkt dieses Prozesses bilden die einzelnen Objekte, die im Prozeß 'Auswahl der Objekte' selektiert wurden und deren Schnittstellenzeiger dem Prozeß in der Variable *pobjects* zur Verfügung gestellt wurden. Diese Objekte werden nach ihrer Zugehörigkeit zu einem Objekttyp zerlegt. Für jeden beteiligten Objekttyp wird ein separater Auswahlsatz erstellt und die entsprechenden Objekte dem Auswahlsatz zugeordnet. Die Eingabe eines eindeutigen Bezeichners für jeden Auswahlsatz ist Aufgabe des Anwenders.

Zwei Besonderheiten beinhaltet die Übernahme der Objekte in den temporären Datenraum. Jedes Objekt bekommt eine neue temporäre Identifikation zugewiesen und jedes Objekt erhält ein neues Attribut *m_lObjectIds* vom Typ $\mathcal{K}_M$ (CIMO_IDENT). Bei der Erzeugung eines Objektes wird dem Attribut das erste Mengenelement zugewiesen, und zwar die Identifikation des persistenten Objektes, welches die Vorlage des Objektes im persistenten Datenraum des Integrationsmoduls darstellt.

4.2.3 Entwicklung eines Interpreters für die Manipulation von Objekten

Zur Bereitstellung einer Eingabeform für den Anwender für die im Abschnitt 3.2.2 auf Seite 38 aufgeführten Operationen wurde eine geeignete Abfragesprache neu entwickelt.

Die Neuentwicklung und der Verzicht sowohl auf kommerzielle Datenbanken als auch auf den Standard der ODMG (Object Database Management Group) begründet sich durch folgende Aspekte (siehe [Heu92] und [Cat93]:

- **statisches Objektmodell**: Das Objektmodell beschreibt die Informationen, die in einer Datenbank gespeichert werden sollen. Hierfür muß der Entwickler Schemabeschreibungen erzeugen, die in der Applikation und in der Datenbank eingebunden werden. Deswegen muß die Struktur der verwendeten Objekte zur Zeit des Compilierens bekannt sein. Da die Objektbeschreibungen dynamisch, also zur Laufzeit des Integrationsmoduls, ermittelt werden, kann das Objektmodell dem Integrationsmodul nicht zur Zeit des Compilierens bekannt sein.

- **Sprachanbindung**: Objektorientierte Datenbanken bieten in der Regel eine objektorientierte Programmiersprache an, um Beziehungen sowie das Verhalten der Objekte zu spezifizieren. Dabei kommen die Programmiersprachen Java, C++ sowie Smalltalk üblicherweise in Frage. Aus Gründen des Laufzeitverhaltens und aus Kompatibilitätsgründen zu verfügbaren Applikationen wurde die Programmiersprache C++ gewählt. C++ beinhaltet nicht die Modellierung von Metaklassen wie beispielsweise Smalltalk, die aber für die Generierung der Objekttypen benötigt werden. Da C++ dieses Konzept nicht unterstützt, kann auch davon ausgegangen werden, daß die objektorientierten Datenbanken, die auf einer C++ Sprachanbindung aufbauen, dieses Konzept in ihrer Abfragesprache nicht berücksichtigen können.

- **Abfragesprache**: Je nachdem unterstützen objektorientierte Datenbanken zum Navigieren durch den Datenbestand eine assoziative Abfragesprache. Diese Abfragesprachen basieren auf SQL (structured query language), unterstützen aber nicht alle SQL-Anweisungen ([Cat93]) wie z.B. die Update-Anweisung. Für die Bearbeitung der übernommenen Objekte nimmt gerade die Update-Anweisung eine Schlüsselrolle ein.

Zu den Aufgaben der Abfragesprache gehören die vollständige Umsetzung der im Abschnitt 3.2.2 auf Seite 38 besprochenen Operationen sowie die Gewährleistung eines Funktionskonzeptes, welches die algorithmische Abbildung von Daten - beispielsweise die Berechnung des euklidischen Abstandes zwischen zwei Punkten - unterstützt und dynamisch erweiterbar ist.

Wortart	Beschreibung	Produktionsregel
Bezeichner	Bezeichner werden als Platzhalter für: - Namen eines Auswahlsatzes - Namen eines Attributes - Namen einer Funktion verwendet	$\{a-zA-Z\_\}\{a-zA-Z0-9\_\}^*$
Zahl	konstanter Ausdruck für einen numerischen Wert	$\{0-9\}\{0-9\}^*(.\{0-9\}^*)?$
Zeichenkette	konstanter Ausdruck für eine Reihe von Zeichen	$'(alle\,druckbaren\,Zeichen)^*'$
Symbol		$+ - *\backslash \% =<> . :, ; \{\}[]!$
Token	weitere Worttrennzeichen	$\backslash n, \backslash t$

Tabelle 4.1: verwendete Wortarten der Abfragesprache

Möglich wird die Implementierung durch das Ausnutzen der speziellen Gegebenheiten, die durch die Verwendung des Integrationsmoduls entstehen. So kann im Integrationsmodul auf eine Einordnung der Objekttypen in Ober- und Unterklassen verzichtet werden, welches ein nicht triviales Problem darstellt [Heu92]. Weiterhin wurde eine vereinfachte Implementierung gewählt, die für jede Abfrage einen neuen Auswahlsatz erzeugt und die darin enthaltenen Objekte wiederholt instanziiert.

Alle Abfragen operieren auf Auswahlsätzen, die sich im temporären Datenraum des Integrationsmoduls befinden. Jeder Auswahlsatz wird über einen eindeutigen Namen, der vom Anwender vergeben wird, identifiziert. Auswahlsätze, die durch das Anwenden von Operationen gewonnen werden, bekommen den Namen des Kommandos zugewiesen, unter dem die Operation als Kommando im Integrationsmodul temporär vorliegt.

Zusätzlich stellen alle Abfragen sicher, daß alle temporären Objekte die Objektidentitäten der persistenten Objekte kennen, aus denen sie hervorgingen. Ein neu erzeugtes temporäres Objekt besitzt eine Referenz zu dem persistenten Objekt, welches das Original im Integrationsmodul darstellt. Kombiniert man zwei temporäre Objekte miteinander, so erhält man ein neues temporäres Objekt, welches die Referenzen von beiden Basisobjekten kennt.

Im folgenden wird die Abfragesprache in der Backus-Naur Notation angegeben. Hierzu werden als erstes die Konstruktionsregeln der verwendeten Wortarten (siehe Tabelle 4.1) und danach die einzelnen Produktionsregeln spezifiziert. Für die Beschreibung wird folgende Syntax verwendet:

Die Erkennung eines Bezeichners kann durch ein vorangestelltes Ausrufezeichen erzwungen werden.

- **Geschweifte Klammern** beschreiben ein Intervall von Zeichen, die für die Bildung des Wortes verwendet werden können.

PROJECTION	FROM	RENAME	ALIAS	EXTEND	JOIN
SELECT	INTERSECT	UNION	INSERT	UPDATE	MINUS
USES	SET	DOUBLE	FLOAT	INTEGER	STRING
LONG	TUPEL	LIST	BOOL	OF	TO
FALSE	TRUE	TYPE	SELSETS	COUNT	SHOW
AND	OR	NOT	CREATE	CREATESET	
HASELEMENTS		REFERENCE		CREATEREFERENCE	

Tabelle 4.2: terminale Schlüsselwörter der Abfragesprache

datentyp	*abfrage*	*bedingung*	*ausdruck*
instanzen	*instanz*	*komponenten*	*komponente*
attributliste	*attribut*	*start*	*paramliste*
funktionsaufruf	*zuweisungen*	*zuweisung*	*memberMethode*
projection	*rename*	*join*	*union*
extend	*select*	*intersect*	*minus*
update	*insert*		

Tabelle 4.3: nichtterminale Schlüsselwörter der Abfragesprache

- **Ein Stern** bedeutet, daß der vorangegangene Ausdruck beliebig (null oder mehrmals) wiederholt werden kann.
- **Ein Plus** bedeutet, daß der vorangegangene Ausdruck mindesten einmal auftreten muß und danach beliebig oft wiederholt werden kann.
- **Ein Fragezeichen** bedeutet, daß der vorangegangene Ausdruck optional ist.
- **Der senkrechte Strich** beschreibt eine Alternative zwischen Ausdrücken.
- **Die runden Klammern** fassen einen Ausdruck zusammen.

Die Abfragesprache verwendet die in den Tabellen 4.2 und 4.3 angegebenen Symbole:

Syntaxbeschreibung der PROJEKTION-Anweisung ..... $\mathcal{P}_k(\wp)$

projection **:**
abfrage '.' PROJECTION '(' *attributliste* ')'.

Diese Produktionsregel führt zur Erzeugung eines auf die in *attributliste* angegebenen Attribute basierenden Tupeltyps und ruft die Methode 'projection' des angegebenen Auswahlsatzes mit dem neu erzeugten Tupeltyp auf. Der implementierte Algorithmus erzeugt den Durchschnitt

zwischen dem zugrundeliegenden Tupeltyp der Objekte und dem angegebenen Tupeltyp. Im Anschluß wird für jedes Objekt $\in$ *auswahlsatz* der Durchschnitt gebildet. Die so manipulierten Objekte werden in dem Ergebnisauswahlsatz gesammelt und dem linken Term der Produktionsregel übergeben.

Syntaxbeschreibung der UMBENENNUNG-Anweisung ..... $\mathcal{R}_{u \to u'}(\wp)$

rename **:**
abfrage '.' RENAME '(' '[' *komponente* ']' ALIAS *Bezeichner* ')'.

Die Implementierung der zu dieser Produktionsregel gehörenden Methode der Klassenbeschreibung des Auswahlsatzes kopiert jedes Objekt des zugrundeliegenden Auswahlsatzes in einen neuen Auswahlsatz. Nach jedem Kopiervorgang wird die zum Tupeltyp des Objektes gehörende Methode 'rename' aufgerufen, welche die Liste mit der spezifizierten Komponente und deren neuen Name übergeben wird. Die Abarbeitung des in *komponente* spezifizierten Zugriffes auf das gewünschte Attribut erfolgt rekursiv, wobei für jeden komplexen Metatyp die entsprechende 'rename'-Methode aufgerufen wird, die als Parameter den restlichen, noch nicht abgearbeiteten Zugriff auf das Attribut erhält. Die 'rename'-Methoden der Metatypen Liste und Menge ruft die 'rename'-Methode des zugrundeliegenden Datentyps für jedes Element der Liste oder Menge auf. Letztendlich bekommt die rename-Methode eines Tupel nur noch den Attributnamen eines zum Tupeltyp gehörenden Attributes. Diesem Attribut wird der als sechster Parameter angegebene Name zugewiesen.

Syntaxbeschreibung der VERBUND-Anweisung ..... $\mathcal{J}(\wp_1, \wp_2)$

join **:**
abfrage '.' JOIN '(' *abfrage* ')' .

Die entsprechende Methode des Auswahlsatzes erzeugt als erstes die Objektbeschreibung des resultierenden Auswahlsatzes. Hierzu werden die zu den angegebenen Auswahlsätzen gehörenden Objektbeschreibungen vereinigt. Anschließend wird für jede Kombination der Objekte des ersten Auswahlsatzes mit den Objekten des zweiten Auswahlsatzes eine neue Objektinstanz in den resultierenden Auswahlsatz übernommen. Die Attributwerte werden vorrangig vom Objekt des ersten Auswahlsatzes der resultierenden Objektinstanz übernommen. Beinhaltet die resultierende Klasseninstanz ein Attribut, welches nicht zur Objektbeschreibung der Objekte im ersten Auswahlsatz gehört, wird der Wert vom Objekt des zweiten Auswahlsatzes extrahiert.

Syntaxbeschreibung der ERWEITERUNG-Anweisung ..... $\mathcal{X}_{<u,s>}(\wp)$

extend **:**
abfrage '.' EXTEND '(' *attributliste* ')' .

Der Algorithmus, der abgearbeitet wird, wenn das Kommando vom Interpreter erkannt wird, erzeugt aus dem fünften Parameter einen Tupeltyp. Der Tupeltyp wird der Methode 'extend' des zugrundeliegenden Auswahlsatzes beim Aufruf übergeben. Die Methode erzeugt einen neuen resultierenden Auswahlsatz mit den gleichen Objekten wie im zugrundeliegenden Auswahlsatz. Zusätzlich wird für jedes Objekt im resultierenden Auswahlsatz die gleichnamige Methode der zum Objekt gehörenden Metatupelklasse aufgerufen, die die Objektbeschreibung mit dem erzeugten Tupeltyp vereinigt.

Syntaxbeschreibung der SELECT-Anweisung ..... $\mathcal{S}_B(\wp)$

select **:**
abfrage '.' SELECT '(' *bedingung* ')'.

Die gleichnamige Methode des Auswahlsatzes erhält im vierten Parameter einen typisierten Baum, dessen Auswertung entscheidet, ob das entsprechende Objekt in den resultierenden Auswahlsatz übernommen wird oder nicht. Die Auswertung des Baumes stellt sicher, daß, wenn die Blätter des Baumes Bezeichnungen von Attributen des Objektes beinhalten, die entsprechenden Attributwerte für die Bestimmung des Wahrheitswertes herangezogen werden. Hierzu werden in einem ersten Durchlauf den Blättern des Baumes die aktuellen Attributwerte eines gerade bearbeiteten Objektes zugewiesen. Anschließend wird, wenn der Baum alle Werte beinhaltet, die Auswertung gestartet. Ergibt die Auswertung den Wahrheitswert TRUE, dann wird das Objekt in den resultierenden Auswahlsatz kopiert. Im Anschluß daran wird das nächste Objekt des zugrundeliegenden Auswahlsatzes betrachtet.

Syntaxbeschreibung der DURCHSCHNITT-Anweisung ..... $\mathcal{A}(\wp_1, \wp_2)$

intersect **:**
abfrage '.' INTERSECT '(' *abfrage* ',' attributliste ')'.

Der Algorithmus dieser Produktionsregel übernimmt nur die Objekte, die in beiden Auswahlsätzen vorhanden sind. Dabei werden für den Vergleich nur die Attribute herangezogen, die in der Produktionsregel angegeben wurden. Der Algorithmus prüft für jedes Objekt des ersten Auswahlsatzes, ob es im zweiten Auswahlsatz vorhanden ist. Ist das Objekt vorhanden, dann wird das Objekt in den resultierenden Auswahlsatz kopiert. Ist es nicht vorhanden, dann wird es nicht übernommen.

Syntaxbeschreibung der DIFFERENZ-Anweisung ..... $\mathcal{M}(\wp_1, \wp_2)$

minus **:**
abfrage '.' MINUS '(' *abfrage* ',' attributliste ')'.

Die Implementierung dieser Produktionsregel übernimmt nur die Objekte in den resultierenden Auswahlsatz, die im ersten Auswahlsatz vorhanden sind, aber nicht im zweiten. Für jedes Objekt im ersten Auswahlsatz wird getestet, ob es im zweiten Auswahlsatz vorhanden ist oder nicht. Im ersten Fall wird es nicht übernommen. Im zweiten Fall wird es übernommen. Das Kriterium für den Objektvergleich bildet die als Parameter angegebene *attributliste*.

Syntaxbeschreibung der VEREINIGUNG-Anweisung ..... $\mathcal{N}(\wp_1, \wp_2)$

union **:**
abfrage '.' UNION '(' *abfrage* ',' attributliste ')'.

Die letzte Produktionsregel, die zu den Mengenoperationen gehört, übernimmt alle Objekte, die entweder im ersten oder im zweiten Auswahlsatz vorhanden sind. Der zugrundeliegende Algorithmus kopiert zunächst alle Objekte des ersten angegebenen Auswahlsatzes in den resultierenden Auswahlsatz. Im Anschluß daran werden die einzelnen Objekte des zweiten Auswahlsatzes sequentiell betrachtet. Ergibt der Objektvergleich, daß ein Objekt des zweiten Auswahlsatzes noch nicht Element des resultierenden Auswahlsatzes ist, dann wird das bestimmte Objekt in den re-

sultierenden Auswahlsatz kopiert. Anderenfalls wird der Algorithmus mit dem nächsten Objekt fortgesetzt.

Syntaxbeschreibung der UPDATE-Anweisung ..... $\mathcal{U}_{u\,{}^{s.u}_{\Psi}\,v}(\wp)$

update **:**
abfrage '.' UPDATE '(' *komponenten* SET *zuweisungen* ')'
| *abfrage* '.' UPDATE '(' *komponenten* USES *komponenten* SET *zuweisungen* ')'.

Das UPDATE-Kommando unterstützt zwei verschiedene Arten, je nachdem, ob in den rechten Termen der Zuweisungen weitere Attribute des Objektes verwendet werden oder nicht. Die hierfür notwendige 'update'-Methode des Auswahlsatzes ruft für jedes Objekt, nachdem das betrachtete Objekt in den resultierenden Auswahlsatz kopiert wurde, die entsprechende 'update'-Methode des zum betrachteten Objekt gehörenden Objekttyps auf. Die Methode des Metatupeltyps benutzt ähnlich wie bei der 'rename'-Methode die gleichnamigen Methoden der komplexen Metaklassen, sobald der Zugriff auf das zu verändernde Attribut über mehrere Hierarchieebenen erfolgt. Bei Listen- und Mengentypen wird die Methode des zugrundeliegenden Listen- oder Mengentyps für jedes Element der Liste oder Menge aufgerufen.

Notwendig ist die Angabe der zusätzlich verwendeten Attribute dadurch, damit eine update-Methode in einer beliebigen Tiefe noch auf den Wert eines übergeordneten Attributes zugreifen kann. Möglich wird dies durch die Implementierung einer Symboltabelle, die alle verwendeten Komponenten eines gerade betrachteten Objektes des zugrundeliegenden Auswahlsatzes beinhaltet und die, noch bevor die update-Methode eines Objektes aufgerufen wird, aktualisiert der entsprechenden 'update'-Methode übergeben wird.

Syntaxbeschreibung der OBJEKTERZEUGUNG-Anweisung

insert **:**
abfrage '.' INSERT '(' ')'
| *abfrage* '.' INSERT '(' UPDATE *komponenten* SET *zuweisungen* ')'
| *abfrage* '.' INSERT '(' UPDATE *komponenten* USES *komponenten* SET *zuweisungen* ')'.

Für das Einfügen eines neuen Objektes in einen bestehenden Auswahlsatz stellt die Objektbeschreibung des Auswahlsatzes die Methode 'addNew' bereit, die entsprechend der zugrundeliegenden Klasseninstanz des Auswahlsatzes eine neue Instanz in den resultierenden Auswahlsatz einfügt. Der Algorithmus, der diesen Produktionsregeln zugrunde liegt, kopiert alle Objekte des zugrundeliegenden Auswahlsatzes in einen resultierenden und ruft anschließend die Auswahlsatzmethode 'addNew' des resultierenden Auswahlsatzes auf. Die letzten beiden Produktionsregeln der Anweisung ändern noch zusätzlich die Attribute der neu angelegten Objektinstanz. Hierfür wird nur für dieses Objekt die 'update'-Metode aufgerufen (siehe UPDATE-Anweisung).

Der Defintion eines neuen Kommandos unterliegt folgender Syntax:

start **:**
abfrage ';'

wobei *abfrage* in eine der folgenden Produktionsregeln abgeleitet werden kann.

abfrage **:**
projection
| *rename*
| *join*
| *extend*
| *select*
| *intersect*
| *minus*
| *insert*
| *update*
| *union*
| *Bezeichner*.

Der Bezeichner muß in diesem Fall ein bestehender Auswahlsatzname des Integrationsmoduls sein.

Die Syntaxbeschreibungen der Operationen benutzen nicht-terminale Symbole, deren Syntax wie folgt definiert wird:

attributliste **:**
attribut
| *attributliste* ',' *attribut*.

Aufgabe des Algorithmus', der nach der Erkennung dieser Produktionsregeln abgearbeitet werden soll, ist es, eine Liste mit den einzelnen Attributen zu erstellen. Die erste Produktionsregel erzeugt dabei die Listeninstanz und trägt das entsprechende, als ersten Parameter angegebene Attribut ein. Die zweite Produktionsregel erweitert die erzeugte Liste um das entsprechende als dritten Parameter angegebene Attribut, wobei das Attribut am Anfang der Liste eingetragen wird.

attribut **:**
Bezeichner ':' *datentyp*.

Der erste Parameter der Produktionsregel stellt einen Attributnamen dar. Der dritte Paramter beinhaltet eine Referenz auf einen gültigen Metaklassentyp. Die zu dieser Produktionsregel gehörende Anweisung erzeugt eine neue Attributinstanz und weist dieser den angegebenen Attributnamen und den angegebenen Datentyp zu.

datentyp **:**
Integer
| Long
| Float
| Double
| Bool
| String
| Tupel Of '<' *attributliste* '>'
| Set Of '[' *datentyp* ']'
| List Of '{' *datentyp* '}'
| Reference To '(' ')'.

Die entsprechenden Anweisungen für diese einzelnen Produktionsregeln erzeugen eine neue Klasseninstanz, basierend auf den angegebenen Namen und liefern die entsprechende Referenz auf die erzeugte Klasseninstanz über das nicht terminale Symbol *datentyp* zurück.

***komponenten* :**
'[' *komponente* ']'
| *komponenten* ',' '[' *komponente* ']'.

Die zu diesen Produktionsregeln gehörenden Anweisungen erzeugen eine Liste von einzelnen Komponenten. Die Anweisung, die zur ersten Produktionsregel gehört, erzeugt die Liste selbst und fügt die angegebene Komponente als erstes Element der Liste ein. Die Anweisung, die zur zweiten Produktionsregel gehört, erweitert die Liste um ein weiteres Element, das wiederum am Anfang der Liste eingetragen wird. Das Element beinhaltet die Referenz auf die angegebene Komponente.

***komponente* :**
Bezeichner
| *komponente* '.' *Bezeichner*.

Die Anweisungen, die diesen Produktionsregeln zugeordnet sind, erzeugen eine Liste bestehend aus Bezeichnungen, die den Zugriff auf ein bestimmtes Attribut des Objektes darstellen. Hierfür erzeugen die Anweisungen der ersten Produktionsregel eine Listeninstanz und tragen den angegebenen Bezeichner in die Liste ein. Die Anweisung der zweiten Produktionsregel erweitert die Liste um den angegebenen Bezeichner.

***bedingung* :**
bedingung AND *bedingung*
| *bedingung* OR *bedingung*
| NOT *bedingung*
| '[' *komponente* ']' HASELEMENTS *instanz*
| '(' *bedingung* ')'
| *ausdruck* '=' *ausdruck*
| *ausdruck* '<' *ausdruck*
| *ausdruck* '>' *ausdruck*
| *ausdruck* '<' '>' *ausdruck*
| *ausdruck* '>' '=' *ausdruck*
| *ausdruck* '<' '=' *ausdruck* .

***ausdruck* :**
TRUE
| FALSE
| '[' *komponente* ']'
| *funktionsaufruf*
| *Zahl*
| *Zeichenkette*
| *ausdruck* '+' *ausdruck*
| *ausdruck* '-' *ausdruck*
| *ausdruck* '*' *ausdruck*
| *ausdruck* '/' *ausdruck*

| *ausdruck* '%' *ausdruck*
| '(' *ausdruck* ')' .

Die einzelnen Produktionsregeln der letzten beiden Ableitungen erzeugen einen binären Baum, bei dem die einzelnen Knoten die Verknüpfungsoperationen zweier Terme und die Blätter die entsprechenden, berechenbaren Werte bzw. Konstanten darstellen. Jeder Knoten besitzt max. zwei Verweise, wobei die Verweise wiederum auf Blätter oder Knoten zeigen können.

Die einzelnen Blätter lassen sich noch spezialisieren in Blätter für die Aufnahme eines Attributwertes, in Blätter für die konstanten Werte und in Blätter für die Aufnahme einer Funktion. Je nach Blattart stehen unterschiedliche Methoden für die Auswertung bereit, um den entsprechenden Wert eines Blattes zu ermitteln.

Die Generierung des Baumes beachtet die Prioritäten der einzelnen Operationen wie beispielsweise Punktoperation vor Strichoperation. Die Semantik der einzelnen Operationen eines Knotens wird immer vom linken Verweis des Knotens bestimmt.

***funktionsaufruf* :**
'#' *Bezeichner* '.' *Bezeichner* '(' ')'
| '#' *Bezeichner* '.' *Bezeichner* '(' *paramliste* ')'.

***paramliste* :**
instanz
| *paramliste* ',' *instanz*.

Für den Aufruf einer Funktion werden zwei Bezeichner verwendet. Der erste Bezeichner entspricht einem Namensbereich, der angibt, zu welchem Komponenten-Interface die gewünschte Funktion gehört. Der zweite Bezeichner differenziert die gewünschte Methode. Da zum Umfang des Integrationsmoduls ein erweiterbares Funktionskonzept gehört, welches wiederum mittels COM dynamisch eingebunden werden kann, ist diese Kennzeichnung einer Funktion notwendig.

Im Abschnitt 6.1.2 auf Seite 161 wird detailliert beschrieben, welche Arbeitsschritte notwendig sind, um ein neues Funktionsinterface zu implementieren, damit das Integrationsmodul diese Methoden benutzen kann.

Aufgabe der Implementierung dieser Produktionsregeln ist es, einen bestimmten Funktionsaufruf in den binären Baum einzubauen. Die entsprechende Struktur ergibt sich aus einem Blattelement des binären Baumes, welches die einzelnen Parameter in einer zusätzlichen Liste aggregiert.

***zuweisungen* :**
zuweisung
| *zuweisungen* ',' *zuweisung*.

Gegenstand des Algorithmus' dieser Produktionsregeln ist es, die einzelnen Zuweisungen in einer Liste zu sammeln und diese Liste über das nicht terminale Symbol *zuweisungen* zurückzugeben. Nachdem die erste Produktionsregel erkannt wurde, wird eine Liste instanziiert und die Referenz der angegebenen Zuweisung in die Liste eingetragen. Je nachdem, ob mehrere Zuweisungen erkannt werden, wird dementsprechend oft die zweite Produktionsregel ausgeführt. Die Anweisungen hierfür fügen die unter *zuweisung* angegebene Referenz in die Liste ein.

zuweisung **:**

'[' *komponente* ']' '=' *instanz*
| '[' *komponente* ']' '+=' *instanz*
| '[' *komponente* ']' '-=' *instanz*
| '[' *komponente* ']' '*=' *instanz*
| '[' *komponente* ']' '\=' *instanz*
| '[' *komponente* ']' '%=' *instanz*
| '[' *komponente* ']' '.' *memberMethode*.

Die einzelnen Anweisungen bauen eine Zuweisung zwischen einer angegebenen Komponente und einem Wert auf. Hierzu wird ein neuer Knoten des binären Baumes erzeugt, der mit dem entsprechenden Zuweisungssymbol und den beiden Referenzen auf die Komponente (linker Teil der Zuweisung) sowie mit dem entsprechenden Wert (rechter Teil der Zuweisung) initialisiert wird.

instanz **:**

ausdruck
| '{' *instanzen* '}'
| '<' *zuweisungen* '>'
| '[' *instanzen* ']'.

Im Gegensatz zu den Abbildungsvorschriften *bedingung* und *ausdruck* stellen diese Produktionsregeln eine Objektinstanz auf, deren Wert sich aus der späteren Auswertung des binären Baumes ergibt. Die erste Produktionsregel erzeugt aus dem angegebenen Ausdruck eine Objektinstanz, basierend auf dem entsprechenden, angegebenen atomaren Datentyp. Die zweite Produktionsregel generiert aus der Liste von Instanzen einen Listentyp, die dritte einen Tupeltyp und die letzte einen Mengentyp. Das nicht terminale Symbol *instanz* beinhaltet nach dem Beenden der entsprechenden Anweisung die Referenz auf die erzeugte Objektinstanz.

instanzen **:**

instanz
| *instanzen* ',' *instanz*.

Die Aufgabe des Algorithmus' dieser beiden Produktionsregeln ist es, eine Liste von Objektinstanzen zu erzeugen. Hierfür wird nach dem Erkennen der ersten Produktionsregel eine neue Liste instanziiert und die angegebene Referenz in die Liste eingefügt. Werden mehrere Instanzen angegeben, dann setzt die Abarbeitung mit der zweiten Produktionsregel fort. Jede erkannte Instanz wird am Anfang der Liste eingefügt.

memberMethode **:**

PROJECTION '(' *attributliste* ')'
| RENAME '(' '[' *komponente* ']' ',' *Bezeichner* ')'
| JOIN '(' *instanz* ')'
| EXTEND '(' *attributliste* ')'
| SELECT '(' *bedingung* ')'
| INTERSECT '(' *instanz* ')'
| MINUS '(' *instanz* ')'
| INSERT '(' ')'
| INSERT '(' *instanz* ')'
| UPDATE '(' *instanz* ')'
| UPDATE '(' USES *komponenten* SET *instanz* ')'

| UNION '(' *instanz* ')'
| COUNT '(' *datentyp* ',' *attribut* ')'
| CREATESET '(' ')'
| CREATEREFERENCE '(' *Bezeichner* ')'
| SET '(' *Bezeichner* ')' .

Die Abarbeitung dieser Produktionsregeln erzeugt einen neuen Eintrag, d.h. ein neues Blatt in dem binären Baum, und gibt dessen Referenz über das nicht terminale Symbol *memberMethode* zurück. Des weiteren fügt der diesen Produktionsregeln zugeordnete Algorithmus die Referenz der angegebenen Parameter in die entsprechende Liste des Blattes ein. Die Abarbeitung des Baumes, die in der übergeordneten 'update'-Methode angestoßen wird, führt zum Aufruf der entsprechenden, gleichnamigen Methode des zugeordneten komplexen Metatyps der Komponente.

Für die Abfrage von Systeminformationen stehen noch folgende Standardabfragen zur Verfügung:

- SHOW SELSETS.
  ⇒ Abfrage der verfügbaren Auswahlsätze der gerade bearbeiteten Datenbank.
- SHOW TYPE FROM *abfrage*.
  ⇒ Abfrage der Typbeschreibung des zugrundeliegenden Auswahlsatzes.
- CREATE *attributliste*.
  ⇒ Erzeugung eines neuen Auswahlsatzes, dessen Objektbeschreibung sich aus einem - aus den einzelnen angegeben Attributen - bestehenden Tupeltyp ergibt und keine Objekte beinhaltet.

Diese Abfragen können nicht mit den anderen Abfragen kombiniert werden.

4.2.4 Erzeugen der Objekte in der externen Applikation

Gegenstand des Prozesses ist es, alle Objekte eines angegebenen Auswahlsatzes in das ausgewählte Dokument zu übernehmen und die Grundlage für den Anwendungsfall 'Wahrung der Konsistenz' zu schaffen.

Die Erzeugung der Objekte erfordert drei Eingaben:

- eindeutiger Name des Auswahlsatzes, welcher die zu übernehmenden Objekte beinhaltet,
- eindeutiger Name des Dokumentes, in dem die neuen Objekte eingefügt werden sollen und
- Name des Objekttypes im Dokument, dem die Objekte des Auswahlsatzes zugeordnet werden sollen.

Für das Bereitstellen der eindeutigen Namen stehen im Integrationsmodul entsprechende Funktionalitäten zur Verfügung. Die möglichen Objekttypen können durch die Analyse der TypeLibrary gefunden werden, die zum entsprechenden Dokument gehört. Objekttypen, die instanziierbar sind, werden in der TypeLibrary besonders gekennzeichnet und verfügen über einen, in der Umgebung der TypeLibrary eindeutigen Namen.

Die Aufgabe der beiden folgenden Teilprozesse ist es, die Objekttypen im Auswahlsatz in den Objekttyp im Dokument abzubilden sowie neue Objekte im Dokument zu instanziieren und anschließend die entsprechenden Attributwerte zu übernehmen. Im Anschluß daran müssen die neu erzeugten Instanzen mit der aktuellen Belegung aller Werte in den persistenten Datenraum des Integrationsmoduls übernommen werden und die Beziehungen zwischen den persistenten Identifikatoren der Objekte des Auswahlsatzes und den persistenten Identifikatoren der erzeugten Instanzen persistent abgelegt werden.

Die Abbildung der Objekttypen wird notwendig, da sich die beteiligten Objektbeschreibungen (Attributname und/oder Attributtyp) voneinander unterscheiden können. Das Einfügen eines zusätzlichen und separaten Prozesses beinhaltet des weiteren noch die Vorteile:

- Auswahl der Attribute, die auch vom Anwender gewünscht werden,
- Verzicht auf alle Attribute, deren Werte nur gelesen werden können und
- Ermöglichen einer komfortablen Eingabe für das Überführen verschiedener Typinformationen.

Die Abbildung der Objekttypen erfolgt durch eine Listenstruktur mit folgendem Aufbau:

1. Name des Attributes in der Objektbeschreibung des Auswahlsatzes.
2. Name des Datentyps des Attributs in der Objektbeschreibung des Auswahlsatzes.
3. Name des Attributes in der Objektbeschreibung des Dokumenttyps.
4. Name des Datentyps des Attributs in der Objektbeschreibung des Dokumenttyps.

Die Voraussetzung für eine erfolgreiche Übernahme ergibt sich aus der Bedingung, daß die angegebenen Datentypen ineinander überführbar sind. Für die Menge der verwendeten Datentypen werden automatisch Konvertierungsmethoden angewendet, sobald die Datentypen nicht den gleichen Wertebereich besitzen. Tabelle 4.4 zeigt die möglichen Konvertierungen an.

Ergibt die Angabe des Zieltyps den Datentyp VARIANT oder DISPATCH, muß der Anwender den gewünschten Typ näher spezifizieren.

Der Datentyp VARIANT läßt nicht die Erkennung des zugrundeliegenden Basistyps zu. Deswegen muß der Anwender die Auswahl durchführen, an welcher Membervariablen (siehe Abschnitt 3.3.2) des Datentyps VARIANT der zu übernehmende Wert abzulegen ist und ob der Wert als Referenz übergeben werden soll.

Die Übernahme eines Attributwertes basierend auf einem Dispatch-Zeiger bedingt, daß eine neue Instanz erzeugt werden muß, deren Referenz auf das entsprechende Dispatch-Interface gerade den aktuellen Attributwert des zu übernehmenden Attributes darstellt. Demnach muß für die Übernahme ein weiterer Typ spezifiziert werden mit den entsprechenden Abbildungsvorschriften. Diese Überlegung verlangt verschachtelte Abbildungsvorschriften, wonach ein Referenztyp eines Attributes im Quellobjekt vorliegt, der in einem Dispatch-Typ eines Attributes im Zielobjekt weitere Abbildungsvorschriften beinhaltet.

Die Übernahme eines Objektes erfolgt schließlich nach folgendem Algorithmus:

	VT_							
	I2	I4	R4	R8	BSTR	BOOL	DISPATCH	VARIANT
Int	true	true	true	true	true	true	false	true
Long	false	true	true	true	true	true	false	true
Float	false	false	true	true	true	false	false	true
Double	false	false	false	true	true	false	false	true
String	false	false	false	false	true	false	false	true
Bool	true	true	false	false	true	true	false	true
Array	false	false	false	false	false	false	false	true
Tupel	false	false	false	false	false	false	true	false
Referenz	false	false	false	false	false	false	true	false

Tabelle 4.4: Gültige Konvertierungen der verwendeten Datentypen

1. Erzeuge eine neue Instanz im Zieldokument basierend auf dem angegebenen Klassennamen.
2. For Each Abbildung In Abbildungsliste
   - Ermittle AttributName, AttributTyp des Quellobjektes der aktuellen Abbildungsvorschrift.
   - Ermittle AttributName, AttributTyp des Zielobjektes der aktuellen Abbildungsvorschrift.
   - Ermittle den Attributwert im zugrundeliegenden Objekt, dessen Name dem ermittelten Attributnamen des Quellobjektes entspricht.
   - Konvertiere den ermittelten Attributwert in den Attributtyp des Zielobjektes.
   - Aufruf der Attributmethode der neu erzeugten Instanz, die den gleichen Namen besitzt wie das Attribut des Zielobjektes, und übergebe der Methode den entsprechenden konvertierten Attributwert.
3. EndFor
4. Beende die Bearbeitung der neu erzeugten Instanz.
5. Übernehme die erzeugte Instanz in den persistenten Datenraum des Integrationsmoduls.
6. Übernehme die Abhängigkeiten des Quellobjektes und der erzeugten Instanz in das Integrationsmodul.

Die Erzeugung einer neuen Instanz im Zieldokument setzt eine weitere Methode voraus, die zum Umfang des neu entwickelten Kommunikationsprotokolls zwischen dem Integrationsmodul und den externen Applikationen gehört. Die Methode besitzt folgende Signatur:

```
HRESULT CreateEmptyInstanz(BSTR ClassName, DISPATCH **pObj)
```

und hat zur Aufgabe, ein neues Objekt mit dem angegebenen Objekttyp zu erzeugen, eine neue, eindeutige und persistente Identifikation dem Objekt zuzuweisen und den Dispatch-Zeiger auf das Objekt zurückzugeben. Die persistente Identifikation des Objektes ist nur im Dokument gültig. Für alle Attribute des Objektes muß die Implementierung der Methode gültige Werte erzeugen und diese den entsprechenden Attributen zuweisen.

Eine weitere Methode, die zum Kommunikationsumfang benötigt wird, beinhaltet die Anweisung 'Beende die Bearbeitung der neu erzeugten Instanz'. Die Methode besitzt folgende Signatur:

HRESULT CloseInstanz(IDispatch **object).

Inhalt der Methode ist es, der externen Applikation nach der Übernahme der Attributwerte die Möglichkeit einzuräumen, notwendige Implementierungen anzugeben. Beispiel hierfür ist die Übernahme des erzeugten Objektes in den persistenten Datenraum der Applikation, wenn die Erzeugung des Objektes erst einmal im temporären Datenraum erfolgte.

Der Schwerpunkt des Algorithmus' liegt in der Anweisung 'Konvertiere den ermittelten Attributwert...'. Diese Anweisung muß die Erzeugung eines Feldes beachten, genauso die Erzeugung von neuen eingenisteten Instanzen, deren Dispatch-Zeiger einen Attributwert darstellt. Für die Erzeugung der eingenisteten Instanzen wird der gleiche Algorithmus, jedoch mit den eingebetteten Abbildungsvorschriften aufgerufen.

Die letzte Anweisung des Algorithmus' wertet die in einer Liste zusammengefaßten persistenten Objektidentifikatoren des aktuellen Quellobjektes aus. Die einzelnen persistenten Objektidentifikatoren wurden bei der Übernahme der Objekte vom persistenten zum temporären Datenraum des Integrationsmoduls und bei der Bearbeitung der Objekte in die Liste eingetragen. Gegenstand der Anweisung ist es, eine neue Relation in ρ zu erzeugen. Für jedes Element der Liste, d.h. für jede persistente Objektidentifikation, wird eine neue Beziehung zur persistenten Identifikation des neu erzeugten Objektes im Zieldokument der Relation ρ generiert.

4.2.5 Spezifikation des Konzeptes für die Wahrung der Konsistenz

Im Abschnitt 3.2.3, Seite 51 wurden die wesentlichen Aspekte für die Integrationsaufgabe 'Wahrung der Konsistenz' vorgestellt. Dieser Abschnitt führt den dort begonnenen Entwurf weiter, indem besonders Designfragen für das Implementationskonzept besprochen werden.

Eine Planungsunterlage im Integrationsmodul wird wie folgt beschrieben:

CIMO_MODELL=$< modell_{id}, application_{id}, modelldaten_{ref}, \Sigma_{id}, zustand >$

mit:
- $modell_{id}$ → Identifikator des CIMO_MODELLS,
- $application_{id}$ → Identifikator der zugrundeliegenden Applikation,
- $modelldaten_{ref}$ → Referenz auf den applikationsspezifischen Datenraum des Dokumentes, die von der Applikation verwendet werden kann,
- Σ_{id} → Menge der überwachten Objekte im persistenten Datenraum des Integrationsmodells,
- $zustand$ → aktueller Zustand des CIMO_MODELLS *stark konsistent, schwach konsistent, schwach inkonsistent, stark inkonsistent.*

Alle CIMO_MODELLe werden hierarisch eingeordnet. Für jede unterstützte Applikation wird ein separater Teilbaum erzeugt, dessen Blätter die einzelnen CIMO_MODELLe bilden. Das Nachrichtenkonzept verwendet einen Manager, der auf jedes CIMO_MODELL zugreifen kann. Das Einfügen einer noch nicht bestehenden Planungsunterlage führt zur Generierung einer neuen Instanz der Klasse CIMO_MODELL. Der Manager erzeugt hierfür eine neue $modell_{id}$ und ist für die korrekte Zuordnung von $application_{id}$ und $modelldaten_{ref}$ verantwortlich. Die restlichen Werte werden wie folgt initialisiert:
$\Sigma_{id} \quad \rightarrow \{o_e\}$
$zustand \rightarrow$ starke Konsistenz.

Aus Performancegründen ist es zweckmäßig, im Integrationsmodul nur die Objekte zu verwalten, die auch von mindestens einem Bearbeiter für die Wahrung der Konsistenz benutzt werden. Aus diesem Grund beinhaltet die Σ_{id} nur ein Element, das Stellvertreterelement (siehe Seite 55). Die starke Konsistenz begründet sich aus der Tatsache, daß eine neu eingefügte Planungsunterlage noch keine Abhängigkeiten zu anderen Objekten besitzen kann.

Der Manager verwaltet noch eine weitere Datenstruktur, die vom Nachrichtenkonzept benötigt wird. Die Datenstruktur wurde schon in Abschnitt 3.2.3, Seite 51 mit dem mathematischen Symbol ρ eingeführt. Sie ist wie folgt definiert:

$$\text{CIMO\_DEPENDENCE} = \{< \sigma_1, \sigma_2, inverse > \mid \begin{array}{l} o_1 =< \sigma_1, s_1, z(s_1) > \in \Sigma_l \wedge \\ o_2 =< \sigma_2, s_2, z(s_2) > \in \Sigma_k \wedge \\ o_2 \text{ ist abhängig von Objekt } o_1 \wedge \\ inverse \in dom(Bool)\} \end{array}$$

ist $inverse$ = TRUE dann ist $< \sigma_1, \sigma_2 > \in \rho \wedge < \sigma_2, \sigma_1 > \in \rho$
ist $inverse$ = FALSE dann ist $< \sigma_1, \sigma_2 > \in \rho \wedge < \sigma_2, \sigma_1 > \notin \rho$.

Die Voreinstellung ist FALSE.

Der Manager muß drei verschiedene Aufgaben realisieren. Zum einen muß er Aktionen bereitstellen, um Abhängigkeiten zwischen den Objekten in den Planungsunterlagen bearbeiten zu können, zum anderen Aktionen, um die Aktualisierung von Objekten zu ermöglichen.

Die Aktualisierung eines Objektes in Σ_{id} kann auf zwei verschiedenen Wegen erfolgen. Zum einem muß der Manager, nachdem eine Planungsunterlage im Integrationsmodul aktualisiert wurde, auch den Zustand aller überwachten Objekte in Σ_{id} überarbeiten. Zum anderen muß der Manager die Aktualisierung eines Objektes durch den Bearbeiter ermöglichen, welches inkonsistent zu einem anderen Objekt ist.

Die dritte Aufgabe ergibt sich aus der Notwendigkeit, die Bearbeiter über den Zustand der einzelnen Planungsunterlagen zu informieren und gegebenenfalls detaillierte Informationen über die Objekte bereitzustellen, die den aktuellen Zustand maßgeblich beinflussen.

Das Nachrichtenkonzept muß demnach eine Reihe von folgenden Teilaufgaben unterstützen:

- Aktualisieren von Modellen.
- Konfiguration der Mengen Σ_{id} und CIMO_DEPENDENCE.
- Ausgabe des aktuellen Zustandes des Nachrichtenkonzeptes.

Teilaufgabe: Aktualisieren von Modellen

Die Aktionen, die der Manager für die Aktualisierung von Objekten bereitstellt, werden durch den Anwender bei folgenden Ereignissen gestartet. Das sind:

- Einbringen eines neuen Freigabestandes, d.h. Aktualisierung eines vollständigen Dokumentes,
- Beheben von inkonsistenten Attributen bezüglich ausgewählter Objekte, d.h. Aktualisieren einer ausgewählten Menge von Objekten mittels den im Integrationsmodul verfügbaren Aktionen.

Die Aktualisierung von Modellen wird erforderlich, wenn ein Eigentümer seine Planungsunterlage auf Grund gegebener Veränderungen anpassen muß. Mögliche Ursachen hierfür sind:

1. Eine Planungsunterlage wird aktualisiert, um auf gestellte externe Anforderungen zu reagieren. Die Ursache für die Veränderungen der betrachteten Planungsunterlage ist nicht in den verwendeten Planungsunterlagen zu suchen.
2. Eine Planungsunterlage wird aktualisiert, um auf aufgetretende Veränderungen in verwendeten Planungsunterlagen zu reagieren.
3. Kombination von beiden.

Generell muß beim Aufstellen der angewendeten Strategie berücksichtigt werden, daß in der überarbeiteten Version der Planungsunterlage noch nicht alle Veränderungen in allen verwendeten Planungsunterlagen berücksichtigt wurden. Der Bearbeiter einer Planungsunterlage muß die notwendigen Entscheidungen selbst treffen, welche veränderten Objekte in den anderen Planungsunterlagen die Objekte der aktualisierten Planungsunterlage beachten. Daraus ergeben sich folgende Situationen:

- Eine Planungsunterlage wird erneut aktualisiert, obwohl in darauf aufbauenden Planungsunterlagen die Aktualisierung der Objekte, die sich aus den aufgetretenden Veränderungen der vorangegangenen Freigabe der betrachteten Planungsunterlage ergaben, noch nicht vollzogen wurde.
- Das Aktualisieren eines Objektes wird nicht notwendig, wenn die Veränderung eines verwendeten Objektes im Toleranzbereich der Lösung liegt.
- Das Aktualisieren eines Objektes wird nicht notwendig, wenn die Veränderung eines verwendeten Objektes im Vorfeld von den beteiligten Anwendern diskutiert wurde. Demnach benutzt der Eigentümer des aktualisierten Objektes schon die Veränderung, die später vom Eigentümer des verwendeten Objektes durchgeführt wird.

Ziel der Teilaufgabe ist es, eine Datenstruktur zu erzeugen und zu verwalten, die Aufschluß über die aufgetretenen Veränderungen gibt und aus der man die sich daraus ergebenden Konsequenzen ableiten kann. Diese Datenstruktur existiert unabhängig von der Datenstruktur CIMO_Dependence, die die Beziehungen zwischen den Objekten persistent ablegt. Die zugrundeliegende Datenstruktur basiert auf dem mathematischen Modell eines gerichteten Graphen mit folgendem Aufbau:

$\mathcal{G} = (\mathcal{V}, \mathcal{E})$ mit $\mathcal{V}$ die Menge alle Knoten.
$\mathcal{E}$ die Menge aller Kanten.

Die Knoten des Graphen besitzen folgenden Aufbau:
$v :=< \sigma, \zeta >\in \mathcal{V}$ mit σ Identifikator des Objektes.
ζ die möglichen Objektzustände $\zeta \in$ {aktualisiert,aktualisierbar,warten}.

Die einzelnen Zustandswerte bedeuten:

aktualisiert: → Das Objekt liegt in einer für den Bearbeiter gültigen Form vor. Das Attribut besagt nicht, daß die Aktualisierung des Objektes alle Abhängigkeiten zu verwendeten Objekten in anderen Planungsunterlagen erfüllt.

aktualisierbar: → Das Objekt kann von dem jeweiligen Bearbeiter aktualisiert werden, da alle Objekte, von denen eine Beziehung zum betrachteten Objekt existiert, den Zustand *aktualisiert* besitzen.

warten: → Das Objekt sollte noch nicht von dem jeweiligen Bearbeiter aktualisiert werden, da mindestens ein Objekt, von dem eine Beziehung zum betrachteten Objekt existiert, noch nicht in den Zustand *aktualisiert* überführt wurde.

In $\mathcal{G}$ besitzen die Kanten neben den beiden Knotenobjekten zwei zusätzliche Attribute. Das erste Attribut λ beschreibt den Zustand der Kante. Der Zustand gibt Auskunft darüber, ob das zur Kante gehörende Kopfobjekt bei der Aktualisierung des Objektes am Ende der betrachteten Kante beachtet wurde (*beachtet*) oder nicht (*unbeachtet*). Das zweite Attribut κ definiert die Art der Veränderung des gleichen Objektes. Die möglichen Werte sind {eingefügt, gelöscht, modifiziert, unverändert, unbekannt}.

Definiert wird eine Kante demnach durch:
$e :=< v, v', \lambda, \kappa >\in \mathcal{E}$ mit v Startknoten der Kante (Kopfelement),
v' Endknoten der Kante,
$\lambda \in \{beachtet, unbeachtet\}$ und
$\kappa \in$ {eingefügt, gelöscht, modifiziert, unverändert, unbekannt}.

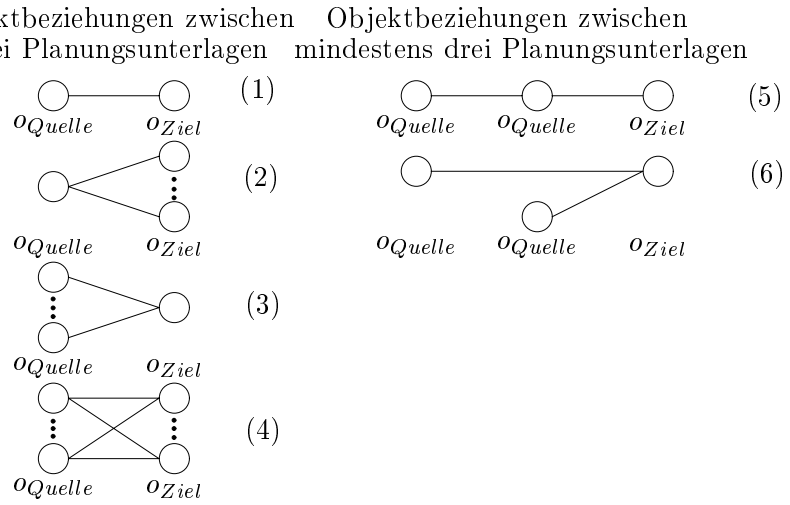
Der Graph $\mathcal{G}$ bildet das Herzstück des Nachrichtenkonzeptes, da er den Zustand der Konsistenz aller Planungsunterlagen im Gesamtsystem dynamisch darstellt. Die Auswertung des Graphen wird in der Teilaufgabe 'Ausgabe des Zustandes des Nachrichtenkonzeptes', beginnend auf Seite 108, erläutert.

Die notwendigen Aktionen dieser Teilaufgabe müssen die veränderten Objekte, die durch eine Aktualisierung einer Planungsunterlage Elemente der zur Planungsunterlage gehörenden Mengen O^D, O^M und O^I sind, sowie deren angegebenen Beziehungen in $\mathcal{G}$ einarbeiten. Jede angegebene Beziehung wurde in CIMO_DEPENDENCE modelliert. Abbildung **??** stellt die möglichen Grundformen dar.

Erfolgt nun die Aktualisierung eines Modells durch das Auslösen des zum Standardumfang von OLE gehörendes Ereignisses 'Aktualisiere Client-Dokument' in der Serveranwendung, erfolgt die Ermittlung der veränderten Objekte in der entsprechenden Ereignismethode nach folgendem Algorithmus:

1. Aktualisiere den veränderten Dokumentinhalt.
2. Instanziiere die drei Mengen O^D, O^I und O^M.

Beziehungen zwischen Objekten



Objektbeziehungen

3. Erzeuge eine Variable *pIObjects*, die den Zugriff auf das Interface CIMO_OBJECTS des Servers ermöglicht.
4. FOR EACH o IN Σ
   - Erstelle aus dem Objekt o ein Objekt o', welches nur die Attribute und deren Werte beinhaltet, die für die Identifizierung des Objektes im Server benötigt werden.
   - Ermittle den Klassennamen des Objektes und speichere den Wert in der Variablen *ClassName*.
   - Ermittle den Zustand des Objektes o' und konvertiere diesen in eine Zeichenkette entsprechend der Produktionsregeln *documentInstance* der Kommandosprache (siehe Seite 83). Die Zeichenkette wird der Variablen *KeyValue* zugewiesen.
   - pDisp = pIObjects->GetObject(ClassName, KeyValue)
   - IF (pDisp = NULL)
     O^D += o.Get_IMOIdent()
     Σ -= o
     ELSE
     Generiere das Objekt o_{Source} basierend auf den Zeiger *pDisp*
     IF (o_{Source} <> o)
     O^M += o.Get_IMOIdent()
     $\Sigma = \Sigma$ - o + o_{Source}
     ENDIF
   - ENDIF
5. ENDFOR
6. Wurde die Ermittlung der neu eingefügten Objekte aktiviert, dann erzeuge die Menge O^I.

Der sechste Schritt bedarf einer weiteren Erläuterung. Die Ermittlung der neu eingefügten Objekte muß separat von einem Anwender angestoßen werden zu einem Zeitpunkt, der vor der Aktualisierung des Dokumentes durch einen anderen Anwender lag (Überwachung). Das Aktivieren der Überwachung für neu eingefügte Objekte führt zur Übernahme aller Objekte einer angegebenen Klasse des Servers sowie zu einem Eintrag in einer zugehörigen Variablen *Eintragungen*, welche Objektklasse betrachtet werden soll. Demnach kann davon ausgegangen werden, daß später, nach einer Aktualisierung des Dokumentes, die zum Dokument gehörende Menge Σ alle Objekte der angegebenen Klasse im vorangegangen Freigabestand beinhaltet.

Für die Überwachung neu eingefügter Objekte stellt das Integrationsmodul ein Ereignis zur Verfügung, dessen Implementierung den Anwender zur Eingabe des gewünschten Dokumentes und des gewünschten Klassentyps auffordert, auf dessen Beschreibung die zu überwachenden Objekte basieren. Durch die Methode

CIMO_OBJECTS::SelectAllObjects(String ClassName);

werden alle Objekte, die auf dem angegebenen Objekttyp basieren und Element des entsprechenden Dokumentes sind, in den persistenten Datenraum des Integrationsmoduls übernommen.

Die Verfeinerung des sechsten Schrittes beinhaltet folgenden Algorithmus:

6.1 FOR EACH ClassName IN Eintragungen

- pIObjects->SelectAllObject(ClassName, &pDispArray)
- Übernehme alle Objekte, die pDispArray referenzieren in einen temporären Datenraum und aggregiere diese in *Objects*
- FOR EACH o IN *Objects*
  IF (o $\notin \Sigma$)
  Σ += o;
  O^I += o;
  ENDIF
- ENDFOR

6.2 ENDFOR

Erfolgt die Aktualisierung des Modells durch das Zuweisen von Attributwerten ausgewählter Objekte mittels der im Integrationsmodul vorhandenen Funktionalität, so sind die veränderten Objekte bekannt. In diesem Fall wird kein spezieller Algorithmus für die Ermittlung der veränderten Objekte notwendig.

Die Manipulation des Graphens $\mathcal{G}$ nach dem Erkennen von veränderten Objekten basiert demnach auf den Mengen O^D, O^M und O^I und auf den modellierten Abhängigkeiten in der Datenstruktur CIMO_DEPENDENCE und erfolgt in zwei Arbeitsschritten. Im ersten Arbeitsschritt werden Bäume generiert, die die Abhängigkeiten eines veränderten Objektes (Wurzelobjekt), welches Element einer der drei Mengen ist, darstellen. In einem zweiten Schritt werden diese Bäume mit dem globalen Graphen $\mathcal{G}$ verschmolzen.

Die Erzeugung der Bäume $\{B_1, B_2, \ldots, B_n\}$ mit $n \in \mathbb{N}$ und $n = card(O^D \cup O^M \cup O^I)$ beachtet alle Objekte, die in irgendeiner Form von dem veränderten Objekt abhängen. Für jede Beziehung, die

während der Bearbeitung in Frage kommt, wird ein neuer Teilbaum erzeugt. Die Konstruktionen der neu angelegten Baumobjekte beinhalten folgende Attributwerte:

- Knotenbelegung eines veränderten Objektes (Wurzelobjekt des Baumes):
  σ $\rightarrow$ Identifikation des betrachteten veränderten Objektes.
  ζ $\rightarrow$ Der Zustand wird auf '*aktuell*' initialisiert.

- Kantenbelegung, dessen Kantenkopf ein verändertes Objekt (Wurzelobjekt) darstellt:
  λ $\rightarrow$ wird mit dem Wert UNBEACHTET initialisiert
  κ $\rightarrow$ zugrundeliegende Art der Veränderung z.B. *gelöscht*.

- Knotenbelegung eines abhängigen Objektes:
  σ $\rightarrow$ Identifikation des betrachteten abhängigen Objektes.
  ζ $\rightarrow$ Der Zustand wird auf '*aktualisierbar*' initialisiert, wenn es sich bei dem Kopfobjekt um ein verändertes Objekt handelt. In allen anderen Fällen wird der Zustand auf *warten* initialisiert.

- Kantenbelegung, dessen Kantenkopf ein abhängiges Objekt darstellt:
  λ $\rightarrow$ wird mit dem Wert *unbeachtet* initialisiert.
  κ $\rightarrow$ wird mit dem Wert *unbekannt* initialisiert.

Für die Generierung des Baumes wird eine rekursive Methode mit folgender Signatur verwendet:

CreateKnoten(Startknoten : CIMO_IDENT, status : ζ, Beachtet : λ, Veränderung : κ),

die ein Knotenobjekt zurückgibt. Die Methode implementiert folgenden Algorithmus:

M.1 Erzeuge ein neues Knotenobjekt v mit den Werten $v =< Startknoten, status >$.

M.2 IF (status = *aktuell*) status = *aktualisierbar*
ELSE IF (status = *aktualisierbar*) status = *warten*
ELSE status = *warten*
ENDIF

M.3 Ermittle alle Objekte des Wertebereiches der Relation ρ, dessen Definitionsobjekt das Objekt mit der als Parameter angegebenen Identifiaktion ist, und speichere die so ermittelten Objektidentifikationen in der Variablen *AbhängigObjekte*. Werte dabei das Attribut *Invert* aus.

M.4 FOR EACH Ident IN *AbhängigObjekte*

- Erzeuge eine neue Kante e
  mit v.AddItem(CreateKnoten(Ident, status, unbeachtet,unbekannt), unbeachtet, Veränderung).

M.5 ENDFOR

M.6 RETURN v

Trifft die Methode während der Erzeugung eines Baumes auf eine *inverse* Abhängigkeitsbeziehung, die ein verändertes Objekt im Wertebereich der Relation ρ enthält, wird eine Nachricht generiert, die das beteiligte Dokument sofort über die aufgetretene Änderung informiert. Ansonsten werden zur Zeit keine *inversen* Abhängigkeiten behandelt (siehe Abschnitt 6, Seite 159).

Die Erzeugung der einzelnen Bäume ergibt unter der Verwendung der obigen vorgestellten Methode folgende Instruktionsfolge:

1. Erzeuge eine Listenvariable *veränderteObjekte*, die alle Wurzelobjekte der generierten Bäume sammelt.
2. For Each Ident In O^M
   - *veränderteObjekte*.AddTail(CreateKnoten(Ident, *aktuell*, *unbeachtet*, *modifiziert*))
3. EndFor
4. For Each Ident In O^I
   - *veränderteObjekte*.AddTail(CreateKnoten(Ident, *aktuell*, *unbeachtet*, *eingefügt*))
5. EndFor
6. For Each Ident In O^D
   - *veränderteObjekte*.AddTail(CreateKnoten(Ident, *aktuell*, *unbeachtet*, *gelöscht*))
7. EndFor

Die Attributwerte der Knoten sowie der Kanten der einzelnen Bäume müssen anschließend in den Graphen $\mathcal{G}$ eingearbeitet werden. Die Überarbeitung muß den aktuellen Zustand der Knoten und Kanten im Graphen beachten. Grund hierfür ist, daß die Objekte der Bäume schon Elemente des Graphens $\mathcal{G}$ sein können.

Das Beachten der möglichen Situationen und der Spezialfälle (siehe Seite 98) schafft folgende klassifizierende Fälle für die Überarbeitung des Graphen $\mathcal{G}$. Ausgangspunkt der Betrachtung bilden die aktualisierten Objekte einer Planungsunterlage, deren Veränderungen in den Knoten und Kanten des Graphens eingearbeitet werden müssen.

- eine Menge von Knoten, die Elemente des Graphen sind und deren zugrundeliegende Objekte bei der Aktualisierung verändert wurden,
- eine Menge von Knoten, die Elemente des Graphen sind und deren zugrundeliegende Objekte bei der Aktualisierung unverändert blieben,
- eine Menge von Knoten, die nicht Element des Graphen sind und deren zugrundeliegenden Objekte bei der Aktualisierung verändert wurden sowie
- eine Menge von Knoten, die nicht Element des Graphen sind und deren zugrundeliegenden Objekte bei der Aktualisierung unverändert blieben.

Zustand eines Objektes im Abhängigkeitsbaum	**alter Zustand des referenzierten Objektes im Graphen**		
	aktualisiert	**aktualisierbar**	**warten**
aktualisisiert	aktualisiert	aktualisiert	warten
aktualisierbar	aktualisierbar	aktualisierbar	warten
warten	warten	warten	warten

Tabelle 4.5: resultierender Zustand eines Objektes im Graphen

Die Bearbeitung der Kopfobjekte einer Kante muß zu einer Überarbeitung der Endobjekte der betrachteten Kante im Graphen führen. Für alle Kanten, die bereits im Graphen enthalten sind, müssen die bestehenden Veränderungen und die Zustände aller Kanten sowie die bestehenden Zustände der Knoten berücksichtigt werden, die diese Kante definieren.

Die Überarbeitung des Graphen beruht letztendlich auf zwei Arbeitsschritten, die wiederholt für jeden Baum und für jede darin enthaltene Kante des Baumes aufgerufen werden. In einem ersten Arbeitsschritt werden die Knoten der Kante überarbeitet und anschließend die Kante selbst. Für die beiden klassifizierenden Fälle, die veränderte Objekte verwenden, beinhalten die neu erzeugten Bäume bereits die Identifikationen der entsprechenden Objekte in ihren Wurzelknoten der Bäume. Die restlichen Objekte können aus der Differenzmenge zwischen der zum aktualisierten Dokument gehörenden Menge Σ und der resultierenden Menge aus der Vereinigung der Mengen O^M, O^D und O^I erzeugt werden. Befinden sich die Objekte im Graphen, so wird der Zustand des Knotens auf *aktuell* gesetzt, welcher das entsprechende Objekt im Graphen darstellt.

Algorithmus für die Überarbeitung des Graphen $\mathcal{G}$

1. Für die Überarbeitung der Knoten des Graphens werden die erzeugten Bäume $B_1, B_2, \ldots$ verwendet. Der verwendete Algorithmus beginnt mit den jeweiligen Wurzelobjekten und überprüft, ob das Wurzelobjekt Element des Graphen $\mathcal{G}$ ist. Wenn ja, wird der Zustand des Objektes entsprechend der Tabelle 4.5 modifiziert. Ist das Wurzelobjekt nicht Element des Graphen, so wird das Objekt entsprechend der aktuellen Belegung im Baum in den Graphen $\mathcal{G}$ übernommen. Im Anschluß daran erfolgt, ausgehend von dem gerade bearbeiteten Kopfobjekt, die Betrachtung der einzelnen Kanten des Baumes.

2. Da jede Kante mindestens zwei Objekte besitzt, wobei das Kopfelement der Kante Element des Graphen sein muß, wird als nächstes das Objekt am Ende der Kante betrachtet. Die Behandlung des Objektes am Ende einer Kante beruht auf den gleichen Anweisungen, wie die Behandlung des Kopfobjektes. Die Behandlung der Kante beginnt mit der Überprüfung, ob die gewünschte Kante in dem Graphen enthalten ist. Wenn ja, werden entsprechend der Tabelle 4.6 die Attribute der Kante modifiziert. Existiert die Kante noch nicht, wird die betrachtete Kante in den Graphen $\mathcal{G}$ übernommen, wobei die Belegung der einzelnen Attribute der Kante beibehalten wird. Der Zustand jeder Kante κ, die betrachtet wird, wird in jedem Fall auf *unbeachtet* gesetzt.

Zustand einer Kante im Abhängigkeitsbaum	**alter Zustand der Kante im Graphen**				
	eingefügt	**manipuliert**	**gelöscht**	**unverändert**	**unbekannt**
eingefügt	n.m.[2]	n.m	n.m	n.m	n.m
manipuliert	manipuliert	manipuliert	n.m.	manipuliert	manipuliert
gelöscht	gelöscht	gelöscht	n.m	gelöscht	gelöscht
unverändert	eingefügt	manipuliert	n.m.	unverändert	unverändert
unbekannt	eingefügt	manipuliert	gelöscht	unverändert	unbekant

Tabelle 4.6: resultierender Zustand einer Kanten im Graphen

Im allgemeinen verändert diese Teilaufgabe nicht den Zustand einer Kante auf *berücksichtigt*. Eine Ausnahme stellt die Aktualisierungsmethode von Objekten dar, die mittels des Integrationsmoduls eine Menge von ausgewählten Objekten aktualisiert. Diese Aktualisierungmethode baut auf den gleichen Arbeitsschritten auf, die in dem Prozeß 'Übernahme von Informationen' zur Definition der Abhängigkeiten führte. Durch diese Arbeitsweise wird sichergestellt, daß alle verwendeten Objekte bei der Aktualisierung benutzt wurden. Befinden sich alle verwendeten Objekte im Zustand *aktuell*, dann können die entsprechenden Kanten von den verwendeten Objekten (Kopfobjekte der Kante) zu den aktualisierten Wurzelobjekten in den Zustand *berücksichtigt* überführt werden.

Im folgenden wird der Ablauf als Struktogramm dargestellt, der bei der Aktualisierung von Objekten im Integrationsmodul abgearbeitet wird. Die Menge O^M ergibt sich aus den Objekten, die Elemente des zur Aktualisierung verwendeten Auswahlsatzes sind. Diese Objektmenge wird für die Aktualisierung verwendet.

1. Ermittle den Zugriff auf die Instanz $\mathcal{G}$ und speichere diese in *p_ Graph*.
2. For Each o In O^M
   - Erzeuge eine Menge $DefObjects = \{\sigma \mid \sigma\rho\eta(o)\}$
   - For Each σ In *DefObjects*
     p_ Graph.SetKantenStatus(σ, o, *beachtet*, *unbekannt*)
   - EndFor
3. EndFor

[2] nicht möglich

Teilaufgabe: Konfiguration der Mengen Σ_{id} und CIMO_DEPENDENCE

Eine Variante, die überwachten Objekte der Menge Σ_{id} hinzuzufügen, erfolgt automatisch während des Prozesses 'Übernahme von Informationen'. Manuell läßt sich die Überwachung auf alle Objekte einer Planungsunterlage ausdehnen. Das vollständige Erkennen von veränderten Objekten in einer Planungsunterlage kann nur manuell aktiviert werden, da der Prozeß 'Übernahme von Informationen' nicht sicherstellen kann, daß alle Objekte eines Modells für die Attributisierung der Zielobjekte verwendet wurden.

Die Standardeinstellung des Nachrichtenkonzepts überwacht ausschließlich die Objekte, die durch den Prozeß 'Übernahme von Informationen' in die Menge Σ_{id} aufgenommen wurden auf folgende Veränderungen:

- Manipulation eines Objektes.
- Löschen eines Objektes.

Das Nachrichtenkonzept operiert auf persistenten Objekten, die bei der Aktualisierung eines bestehenden oder bei der Erzeugung eines neuen Objektes kopiert wurden. Die Kopien werden in die Menge Σ der entsprechenden Instanz von CIMO_MODELL eingefügt. Das heißt, alle Objekte, die als Informationsträger auftreten sowie alle Objekte, die die ermittelten Informationen verwenden, sind in der entsprechenden Menge Σ enthalten und können für die Ermittlung von Veränderungen verwendet werden. Zusätzlich werden neue Tupel, die die Identifizierung zweier abhängiger Objekte in Beziehung setzen, in der Instanz der Klasse CIMO_DEPENDENCE erzeugt, sobald ein Objekt (Quelle) eine Information dem anderen Objekt (Ziel) zur Verfügung stellt.

Die manuelle Konfiguration der Menge Σ_{id} basiert auf zwei Arbeitsschritten:

- Spezifikation von Objekten in einer fremden Planungsunterlage, die überwacht werden sollen.
- Spezifikation von Objekten der eigenen Planungsunterlage, die von einer Veränderung der überwachten Objekte betroffen sind.

Die manuelle Konfiguration erfordert die Angabe einer Menge, die die Objekte beinhalten, die miteinander in Beziehung zu setzen sind. Das Integrationsmodul stellt hierfür drei Kommandos zur Verfügung:

CREATE DEPENDENCE FROM *auswahlsatz*;
CREATE DEPENDENCE FROM $modell_{id_1}$, *classname* TO $modell_{id_2}$;
CREATE DEPENDENCE INTERACTIV FROM $modell_{id_1}$ TO $modell_{id_2}$;.

Die erste Möglichkeit erwartet einen Auswahlsatz, dessen Datentyp ein Tupeltyp $\mathcal{K}_T(\{a_1, a_2\})$, bestehend aus zwei Attributen, ist. Das erste Attribut $a_1 =< def_o$,CIMO_IDENT, $normal >$ beschreibt das Definitionsobjekt durch seine Identifikation.
Das zweite Attribut $a_2 =< bild_o$,CIMO_IDENT, $normal >$ enthält die Identifikation des Objektes des Wertebereiches der Relation.
Für jedes Objekt im angegebenen Auswahlsatz wird ein neues Element in der globalen Datenstruktur CIMO_DEPENDENCE erzeugt $s(\mathcal{K}_T.def_o)\rho s(\mathcal{K}_T.bild_o)$.

Die zweite Möglichkeit übernimmt alle Objekte der unter $modell_{id_1}$ angegebenen Planungsunterlage, welche Instanzen der angegebenen Klasse (*classname*) sind. Jedes Objekt wird in die Komponente Σ_{id_1} der dazugehörenden Instanz der Klassse CIMO_MODELL eingefügt. Im Anschluß daran erzeugt das Kommando für jedes übernommene Objekt ein neues Tupel von der Datenstruktur CIMO_DEPENDENCE, mit dem übernommenen Objekt als Defintitionswert und dem Stellvertreterobjekt der Planungsunterlage, die unter $modell_{id_2}$ angegeben wurde, als Wertebereichswert.

Die dritte Möglichkeit erzeugt die in Beziehung stehenden Objekte interaktiv, indem abwechselnd die Objekte des Definitionsbereiches sowie die Objekte des Wertebereiches ausgewählt werden. Zusätzlich ist die Angabe des Stellvertreter-Objekts der Planungsunterlage $modell_{id_2}$ möglich. Werden mehrere Objekte in einem Bearbeitungsschritt für den Definitionsbereich oder den Wertebereich ausgewählt, dann wird jedes Objekt des Definitionsbereiches mit jedem Objekt des Wertebereiches kombiniert. Für jede Kombination wird ein separater Tupel in die Datenstruktur CIMO_DEPENDENCE eingefügt.

Das Erkennen von neu eingefügten Objekten nimmt in diesem Zusammenhang eine besondere Stellung ein. Für das Erkennen dieser Veränderung stellt die Menge Σ^*_{id} die Ausgangsbasis dar. Außerdem muß die Aktion, die die Mengen Σ_{id} und Σ^*_{id} erzeugt, sicherstellen, daß kein Objekt für die Bildung der Mengen vergessen wird. Aus diesem Grund kann das Erkennen von neu eingefügten Objekten in eine Planungsunterlage nur durch eine spezielle Aktion realisiert werden. Die Synatx des Kommandos lautet:

ACTIVATE DEPENDENCE INSERT FROM $modell_{id_1}$, *classname* TO $modell_{id_2}$;

Dieses Kommando realisiert dieselben Arbeitsschritte wie die zweite Möglichkeit, manuelle Beziehungen zwischen den Objekten zu definieren. Zusätzlich aktiviert dieses Kommando die Bildung der Menge O^I in der unter $modell_{id_1}$ angegebenen Planungsunterlage.

Das letzte Kommando ermöglicht die Defintion der inversen Abhängigkeit zweier Objekte. Hierzu müssen die Objekte einer Planungsunterlage angegeben werden, die durch eine Veränderung ihrer Attributisierung eine Aktualisierung bei den Objekten auslöst, die für die Definition der veränderten Objekte verwendet wurden. Das Kommando kann nur auf bestehende Tupel der Datenstruktur CIMO_DEPENDENCE angewendet werden. Mit der Aktion:

INVERSE DEPENDENCE FROM $modell_{id_1}, objekttyp_1$ TO $modell_{id_2}, objekttyp_2$;

werden alle Tupel der Datenstruktur selektiert, die die folgende Relation erfüllen:

$$\begin{aligned}\rho_{Inverse} = \{ < \sigma_1, \sigma_2, invert > | < \sigma_1, \sigma_2, false > \in \rho \wedge \\ < \sigma_1, s_1, z(s_1) > \in \Sigma_{id_1} \wedge s_1 = objekttyp_1 \wedge \\ < \sigma_2, s_2, z(s_2) > \in \Sigma_{id_2} \wedge s_2 = objekttyp_2 \}.\end{aligned}$$

Für alle selektierten Tupel wird der Wert der Variablen *invert* auf den Wert TRUE gesetzt. Die Angaben $objekttyp_1$ und $objekttyp_2$ sind optional. Wird auf die Angabe des Objekttyps verzichtet, dann wird auch die entsprechende Bedingung für die Erzeugung der Relation $\rho_{Inverse}$ nicht ausgewertet.

Das Einfügen einer Kopie in eine Menge Σ_{id} kann nur erfolgen, wenn das betrachtete Objekt noch nicht Element der Menge ist. Für den Vergleich zweier Objekte einer Planungsunterlage wird die Objektidentität (siehe Abschnitt 3.4 auf Seite 64) der Objekte in der entsprechenden

Planungsunterlage verwendet. Die Durchführbarkeit des Vergleiches führte zu der getroffenen Voraussetzung an die Applikationen, eine persistente Identifikation für jedes Objekt zu verwalten.

Teilaufgabe: Ausgabe des Zustandes des Nachrichtenkonzeptes

Die Ausgabeschnittstelle des Nachrichtenkonzeptes basiert auf dem Graphen $\mathcal{G}$. Alle Attribute, bis auf den Zustand einer Kante, werden vom Manager des vorgestellten Nachrichtenkonzeptes verwaltet. Da der Manager nicht die Entscheidung fällen kann, ob ein Anwender eine Beziehung beachtet hat oder nicht, muß er die Änderung des Zustandes einer Kante auf *beachtet* den Anwendern überlassen.

Für die Ausgabeschnittstelle des Nachrichtenkonzeptes steht eine Dialogbox zur Verfügung. Die Dialogbox gibt in Abhängigkeit einer spezifizierten Planungsunterlage:

- den Zustand der Planungsunterlage,
- die Anzahl der aktualisierbaren Objekte,
- die Anzahl der Objekte, die in dem Zustand *warten* sind und
- die Anzahl der aktualisierten Objekte, deren Beziehungen zu den verwendeten Objekten noch nicht beachtet wurden

an.

Der Zustand einer Planungsunterlage ergibt sich aus dem Auswerten der ermittelten Kardinalitäten. Eine Planungsunterlage ist demnach:

1. **stark konsistent,**
   wenn die Anzahl der aktualisierbaren und wartenden Objekte gleich Null ist und wenn es keine Beziehung mehr gibt, die beachtet werden muß.
2. **schwach konsistent,**
   wenn die Anzahl der wartenden Objekte gleich Null ist. Die Anzahl der aktualisierbaren Objekte oder die Anzahl der zu aktualisierenden Beziehungen sind größer Null.
3. **schwach inkonsistent,**
   wenn die Anzahl der wartenden Objekte der Planungsunterlagen größer Null ist. Die Anzahl der aktualisierbaren Objekte und die Anzahl der Beziehungen sind gleich Null.
4. **stark inkonsistent,**
   sonst.

Die Bestimmung der Mengen erfolgt durch das gezielte Selektieren in dem Graphen $\mathcal{G}$.

Im Abschnitt 3.2.3 auf Seite 51 wurde schon hergeleitet, daß der Anwender für die Aktualisierung einer Planungsunterlage noch genauere Informationen wünscht. Das Nachrichtenkonzept stellt hierfür dem Anwender eine Übersicht über die Objekte, die in der Planungsunterlage und im

Graphen enthalten sind, sowie über deren Zustände zusammen. Durch das Auswählen eines Objektes können weitere Informationen angezeigt oder vom Objekt abhängige Aktionen gestartet werden.

Für ein ausgewähltes Objekt stehen folgende weitere Informationen zur Verfügung:

- Liste von Objekten, von denen eine Kante zu dem ausgewählten Objekt existiert (alle Vorgänger).
- Liste von Objekten, die eine Kante zum ausgewählten Objekt besitzen (alle Nachfolger).

Für jedes Objekt werden die Attributwerte des Knotens und der entsprechenden Kanten ausgegeben. Die Darstellung unterstützt das Verfolgen aller Abhängigkeiten eines Objektes in beliebiger Tiefe. Das heißt, durch das Auswählen eines Vorgängerobjektes können wiederum alle Vorgänger ermittelt werden.

Die Aktionen ermöglichen den Anwendern, den Zustand eines Objektes sowie den Zustand einer Kante zu modifizieren. Ein Objekt kann in den Zustand *aktualisiert* überführt werden, wenn der aktuelle Zustand des Objektes *aktualisierbar* ist. Damit beachtet das Nachrichtenkonzept den Sonderfall (4) auf Seite 98. Die zweite Aktion erlaubt den Anwendern, den Zustand einer Kante auf *beachtet* zu setzen, wenn die beiden beteiligten Objekte den Zustand *aktualisiert* besitzen. Damit überläßt das Nachrichtenkonzept dem Anwender die Entscheidung, was er bei der aktualisierten Planungsunterlage beachtet hat oder was nicht.

Bis jetzt wurde nur die Verwendung des Graphen in Hinblick auf eine Manipulation und dem Hinzufügen von Kanten und Knoten betrachtet sowie die Möglichkeiten der Auswertung des Graphen dargestellt. Natürlich muß der Graph auch einer Bereinigung unterzogen werden, das heißt, bestimmte Knoten und Kanten müssen aus dem Graphen gelöscht werden, wenn die Objekte und die Kanten von den jeweiligen Bearbeitern abgearbeitet wurden.

Unkritisch ist das Löschen aller Kanten und Knoten, wenn sich alle Planungsunterlagen im Zustand STARK KONSISTENT befinden. Da sich dieser Fall normalerweise erst zum Ende des Projektes einstellt, würde der Graph unüberschaubare Dimensionen annehmen.

Das Löschen einer Kante, wenn sie in den Zustand *beachtet* gesetzt wird, würde den Sonderfall (3) auf Seite 98 nicht beachten. Betrachtet man drei Objekte o_1, o_2, o_3 mit den Kanten $o_1 \rightarrow o_2 \rightarrow o_3$ und wird das Objekt o_1 manipuliert, dann würde nach der Aktualisierung von o_2 und der anschließenden Zustandsänderung der Kante $o_1 \rightarrow o_2$ auf *beachtet* diese Kante aus dem Graphen entfernt werden. Der Manager kann den entsprechenden Eigentümer des Objektes o_3 nicht mehr über die Veränderung des Objektes o_1 informieren.

Zweckmäßig für das Bereinigen des Graphen ist es, die zu erwartende Struktur zu benutzen. Da die Kanten Objektabhängigkeiten darstellen und nicht jedes Objekt mit jedem Objekt in Beziehung steht, ist eher ein nicht zusammenhängender Graph zu erwarten. Damit kann das Bereinigen eines Graphen durch die Betrachtung eines zusammenhängenden Teilgraphen erfolgen. Ein zusammenhängender Teilgraph kann entfernt werden, wenn alle Kanten des Teilgraphen den Zustand *beachtet* und alle Knoten den Zustand *aktualisiert* des Teilgraphen besitzen.

Die grundlegende Voraussetzung des Nachrichtenkonzeptes ist, daß der Graph $\mathcal{G}$ zyklenfrei ist. Prinzipiell ist es jedoch möglich, Zyklen zwischen verschiedenen Objekten zu modellieren. Diese

Zyklen können nur durch die manuelle Konfiguration der Relation CIMO_DEPENDENCE auftreten. Erkannt werden können diese Zyklen durch einen Algorithmus (siehe [Jun90], [Bod95] und [Freu98]), der nach der Überarbeitung des Graphens automatisch aufgerufen wird.

Grundsätzlich verstoßen Abhängigkeiten, die zu einem Zyklus führen, gegen festzulegende Regelungen zwischen den am Projekt beteiligten Fachplanern. Die Regeln legen eindeutig eine Rangordnung fest, wonach ein Fachplaner vorgabeberechtigt ist und ein anderer diese Vorgaben in seiner Planungsunterlage verwenden muß. Erschwerend jedoch ist, daß diese Regeln nicht üblicherweise so kompromißlos angewendet werden. Als möglicher Ausweg kann den Anwendern eine Kompromißlösung angeboten werden, die die Abbildung solcher zu Zyklen führenden Abhängigkeiten zwischen Objekten vermeidet und trotzdem die Erzeugung der gewünschten Nachrichten durch den Manager zuläßt. Hierzu wird die Abbildung auf das Stellvertreterobjekt der Planungsunterlage vorgeschlagen.

Der letzte Punkt behandelt die Zeitpunkte, an denen die Operationen des Managers ausgelöst werden. Die Bereinigung des Graphen erfolgt automatisch, wenn ein Anwender eine Menge von Kanten auf den Zustand *beachtet* gesetzt hat. Die Erweiterung des Graphen erfolgt nach dem Aktualisieren einer gesamten Planungsunterlage oder nach dem Aktualisieren bestimmter Objekte durch entsprechende Aktionen des Integrationsmoduls.

4.3 Spezifikation der externen Schnittstellen

Dieser Abschnitt faßt noch einmal die Methoden zusammen, die für die Kommunikation zwischen dem Integrationsmodul und den externen Applikationen verwendet werden und an verschiedenen Stellen in dieser Arbeit eingeführt wurden.

Zweckmäßig ist die Zuordnung der einzelnen Kommunikationsmethoden zu drei Schnittstellen. Die erste Schnittstelle beinhaltet die Serveraktionen, die zweite Schnittstelle die einzelnen Aktionen auf den Objekten. Die dritte Schnittstelle wird für die iterative Abarbeitung einer Menge von Objekten benötigt. Die einzelnen Schnittstellen unterliegen dem in Abbildung 4.6 dargestellten Aufbau.

Die zweite Gruppe von Kommunikationsmethoden, die eine Applikation unterstützen muß, wenn sie in das Integrationsmodul integriert werden soll, ermöglicht das Einbetten und das Linken eines Server-Dokumentes in ein Client-Dokument (OLE) sowie den Zugriff auf die Objekte mittels ActiveX-Automation.

Für die Spezifikation der Automation-Schnittstelle der einzelnen Objekte gelten folgende Empfehlungen: Flache Objektbeschreibungen, die die Beziehungen nicht mehrmals ineinander verschachteln, erleichtern die Übernahme von referenzierten Objekte. Des weiteren sollte bei der Spezifikation auf Containerklassen verzichtet werden, da diese die Übernahme kompliziert gestalten. Die Verwendung eines Arrays erfüllt den gleichen Zweck. Abschnitt 6.2.2 auf Seite 164 stellt eine Möglichkeit vor, wie Containerklassen im Integrationsmodul verwendet werden können.

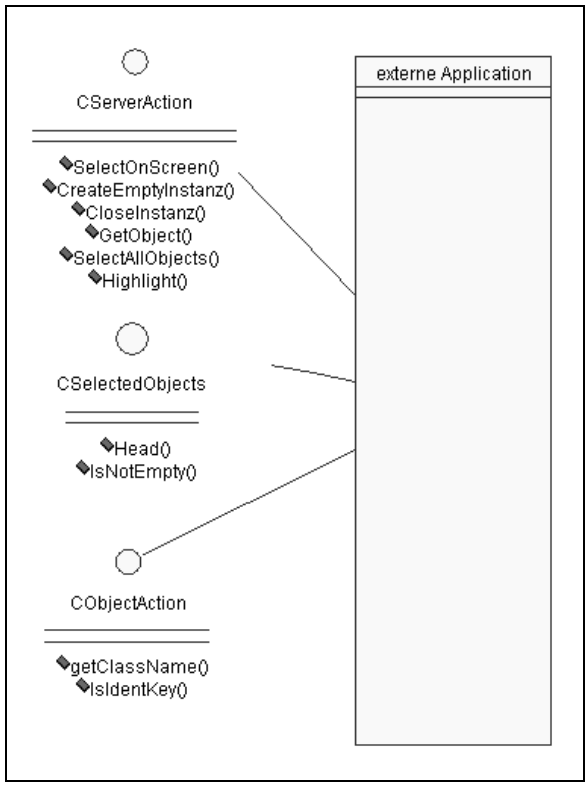


Abbildung 4.6: Schnittstelle für die Kommunikation

5 Fallbeispiel

Der vorgestellte Ansatz soll nun an repräsentativen Beispielen getestet werden. Ausgangsbasis bildet ein CAD-Plan (siehe Seite 116), der den Grundriß des Erdgeschosses von einem Wohnhaus darstellt. Hierzu sollen folgende Integrationsaufgaben betrachtet werden:

- Mengenermittlung der Wände, Fenster, Türen und Stürze aus einem CAD-Plan,
- Generierung von Objekten für die statische Bemessung von Unterzügen,
- Löschen eines Fensters und die darauf aufbauende Aktualisierung.

Die Fallbeispiele beinhalten das Ziel, die Flexibilität des Integrationskonzeptes sowie die Anforderungen eines Anwenders an Integrationssoftware zu verdeutlichen. Das erste Beispiel dokumentiert die Übernahme von Mengendaten, obwohl diese Daten im Modell entweder nicht in der erforderlichen Form oder gar nicht vorhanden sind. Das zweite Beispiel erläutert verschiedene Varianten der Objekterzeugung. Ausschlaggebend hierbei ist die entsprechende Entscheidung des steuernden Anwenders. Das letzte Beispiel beschreibt den Einsatz von Software für die rechentechnische Überwachung der Konsistenz verschiedener Planungsunterlagen (Dokumente).

5.1 Erläuterung der Aufgabenstellung: Übernahme von Daten

Ausgangsbasis für die Mengenermittlung bilden die Objekte des CAD-Plans sowie die gewünschte Mengenermittlung entsprechend der VOB-Teil C. In den folgenden drei Abschnitten werden die Strukturbeschreibung der Objekte eines CAD-Systems und die Strukturbeschreibung der AVA-Objekte wiedergegeben sowie die notwendigen und anzuwendenden Regeln der VOB-Teil C.

5.1.1 Strukturbeschreibung der Objekte eines CAD-Systems

Für die Strukturbeschreibung der Objekte wurde die Automation-Schnittstelle des Architekturprogramms ARCON verwendet. Demnach besitzt ein Wandobjekt in diesem CAD-System folgenden Aufbau:

$wall = \mathcal{K}_T(\{$<AverageArea,Float,normal>, ⇒ die mittlere Wandfläche
<AverageAreaFormula,String,normal>, ⇒ Formel für die mittlere Wandfläche
<AverageLength,Float,normal>, ⇒ die mittlere Wandlänge
<Height,Float,normal>, ⇒ die Höhe der Wand bis Unterkante Decke
<ID,Long,key>, ⇒ eine eindeutige Kennung
<Thickness,Float,normal>, ⇒ die Dicke der Wand
<Type,Long,normal>, ⇒ der in ARCON definierte Wandtyp
<Doors,$\mathcal{K}_{Ref}$({Door}),normal>, ⇒ Liste mit assoziierten Türen
<Windows,$\mathcal{K}_{Ref}$({Window}),normal>, ⇒ Liste mit assoziierten Fenstern
}).

Der Objekttyp *wall* referenziert Objekte. Die Objektbeschreibung der Tür lautet:

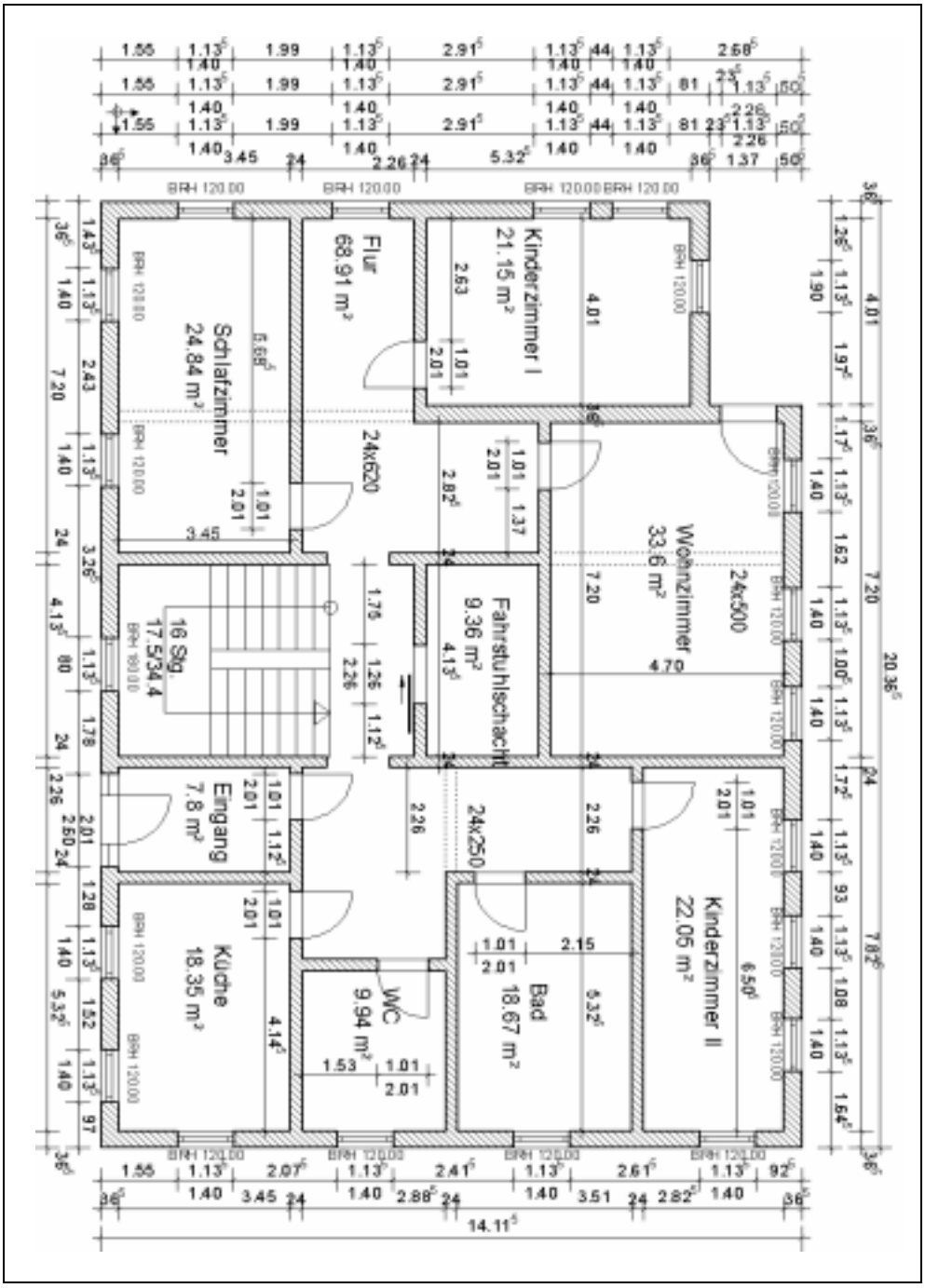


Abbildung 5.1: Erdgeschoß

Door = $\mathcal{K}_T$({<Area,Float,normal>, ⇒ Fläche der Tür in Quadratmetern
<AreaFormula,String,normal>, ⇒ Formel zur Berechnung der Türfläche
<Height,Float,normal>, ⇒ Höhe der Tür
<HungLeft,Bool,normal>, ⇒ Scharnier auf der linken Seite?
<ID,Long,key>, ⇒ Identifikation des Objektes in der Applikation
<LeftWingAngle,Float,normal>, ⇒ Öffnungswinkel des linken Flügels[rad]
<LeftWingType,Long,normal>, ⇒ Art des linken Türflügels
<OpensInwards,Bool,normal>, ⇒ Tür öffnet nach innen?
<Remark,String,normal>, ⇒ Bemerkung zur Tür
<RightWingAngle,Float,normal>, ⇒ Öffnungswinkel des rechten Flügels[rad]
<RightWingType,Long,normal>, ⇒ Art des rechten Türflügels
<Type,Long,normal>, ⇒ der in ARCON definierte Türtyp
<Width,Float,normal> ⇒ Breite der Tür
}).

Ein Fenster besitzt in ARCON folgende Attribute:

Window = $\mathcal{K}_T$({<Area,Float,normal>, ⇒ Fläche des Fensters in Quadratmetern
<AreaFormula,String,normal>, ⇒ Formel zur Berechnung der Fensterfläche
<Height,Float,normal>, ⇒ Höhe des Fensters
<ID,Long,key>, ⇒ Identifikation des Objektes in der Applikation
<LeftHung,Bool,normal>, ⇒ Fenster links angeschlagen?
<OpensInwards,Bool,normal, ⇒ Fenster nach innen öffnend?
<ParapetHeight,Float,normal>, ⇒ Brüstungshöhe
<Remark,String,normal>, ⇒ Bemerkung zum Fenster
<Type,Long,normal>, ⇒ der in ARCON definierte Fenstertyp
<Width,Float,normal> ⇒ Breite des Fensters
}).

Als nächstes folgt die Strukturbeschreibung einer Decke in ARCON. Das Deckenobjekt wird für die Ermittlung der Wandmengen benötigt. Eine Decke besitzt in ARCON folgende Attribute:

Ceiling = $\mathcal{K}_T$({<ID,Long,key>, ⇒ Identifikation des Objektes in der Applikation
<Openings,$\mathcal{K}_{Ref}$({Ceiling}),normal>, ⇒ Öffnungen in der Decke
<Thickness,Float,normal>, ⇒ die Dicke der Decke
}).

Als letztes wird nun der Objekttyp eines Unterzuges vorgestellt. ARCON modelliert Unterzüge und Überzüge in ein und derselben Datenstruktur. Der Objekttyp hat folgenden Aufbau:

UnterÜberzug = $\mathcal{K}_T$({<Height,Float,normal>, ⇒ Höhe des Zuges
<ID,Long,key>, ⇒ Identifikation des Objektes in der Applikation
<Length,Float,normal>, ⇒ Länge des Zuges
<Thickness,Float,normal>, ⇒ Dicke des Zuges
<Position,$\mathcal{K}_S$({Float}),normal>, ⇒ Koordinaten der Eckpunkte 2D
<Ueberzug,Bool, normal> ⇒ Objekt ist ein Überzug?
}).

Die hier vorgestellten Objekttypen sind nicht vollständig entsprechend der ActiveX-Documentation von ARCON. Beispielsweise unterstützen alle diese Typen noch ein zusätzliches Attribut *Texture*,

Kennzeichnen	Satzzeichen
00	Eröffnungssatz
11	Aufmaßsatz
99	Schlußsatz

Tabelle 5.1: Satzarten der DA11-Schnittstelle

welches die Darstellung der Objekte steuert. Aus Gründen der Übersichtlichkeit soll eine vereinfachte Beschreibung an dieser Stelle ausreichen.

5.1.2 Strukturbeschreibung der Objekte eines AVA-Systems

Für die Strukturbeschreibung dieser Objekte soll der Standard der GAEB[1]-Schnittstelle DA11 benutzt werden. Ausgangspunkt der Überlegung ist, daß dieses Datenaustauschformat für die Übertragung der geplanten Mengen verwendet wird und somit alle hierfür relevanten und notwendigen Informationen beinhaltet. Des weiteren kann davon ausgegangen werden, daß die Struktur für die in einem AVA-System verwendeten Objekttypen mit der Struktur des Datenaustauschformates weitestgehend übereinstimmt.

Das entsprechende Datenaustauschfile ist zeilenorientiert aufgebaut, wobei jede Zeile aus 80 Zeichen besteht. Es unterstützt drei Satzarten, deren Kennzeichnung in den ersten beiden Bytes einer Zeile erfolgt. Tabelle 5.1 stellt die einzelnen Satzzeichen sowie deren Kennzeichnung vor. In diesem Zusammenhang spielt die zweite Satzart die dominante Rolle, da dieser Satz die ermittelten Mengen beinhaltet. Das Aufsatzmaß setzt sich aus 13 Attributen zusammen, deren Werte feste Positionen in der Datei besitzen. Tabelle 5.2 stellt den Inhalt und die Position der einzelnen Attribute vor.

Für die Übernahme der Daten werden nicht alle 13 Attribute benötigt. Wichtig in diesem Zusammenhang sind:

$position = \mathcal{K}_T(\{$<Z1,String,key>, ⇒ Z1-Nummer
<Z2,String,key>, ⇒ Z2-Nummer
<Position,String,key>, ⇒ Positionsnummer
<Index,String,key>, ⇒ Index zur Position
<Erläuterung,String,normal>, ⇒ Erläuterung
<Mengenansatz,$\mathcal{K}_S(\{MAnsatz\})$,normal> ⇒ Mengenansatz des Bauteils
$\}).$

Die Struktur aggregiert einen weiteren Datentyp. Die Definition des Datentyps unterliegt folgendem Aufbau:

$MAnsatz = \mathcal{K}_T(\{$<Erläuterung,String,normal>, ⇒ Erläuterung
<Vorzeichen, String,normal>, ⇒ Vorzeichen des Faktors
<Faktor,Float,normal>, ⇒ Faktor
<Ansatz,String,normal> ⇒ Berechnungsformel
$\}).$

[1]GAEB ⇒ Gemeinsamer Ausschuß Elektronik im Bauwesen

Spalte	Inhalt
1 -2	Satzkennzeichen
3 -11	Ordnungszahl(Z1,Z2,Pos,Index)
12 -12	Zwischensummenindex
13 -13	Kenzeichen, beinhaltet diese Spalte einen Stern, dann enthalten die nächsten 5 Spalten einen Kommentar
14 -22	Erläuterung
23 -23	Vorzeichen für Faktor
24 -29	Faktor(Anzahl)
30 -31	Formelnummer
32 -69	Ansatz
70 -73	Blatt-Nummer
74 -74	Zeile
75 -75	Zeilenindex
76 -80	zur besonderen Verwendung

Tabelle 5.2: Beschreibung des Aufmaßsatzes [Schnei94]

5.1.3 Anzuwendende Regeln für die Mengenübernahme

Traditionell ermittelt der AVA-Anwender die einzelnen Mengen bauteilorientiert, wobei der Schwerpunkt auf der Darstellung eines nachvollziehbaren Berechnungsweges liegt. Für jedes Bauteil wird eine Position angelegt, die die Menge des Bauteils angibt. Gleichartige Bauteile werden später zusammengefaßt, wobei die Art und Weise der Zusammenfassung abhängig von dem jeweiligen Bearbeiter ist. Aufgabe des Integrationsprozesses kann es nur sein, die jeweiligen Ansätze für die Berechnung der Mengen in das AVA-System einzufügen. Hierbei sind jedoch die Regeln für die Ermittlung der Mengen zu beachten.

Gemäß der VOB-C basiert die Mengenermittlung der Wände auf der Basis ihrer Flächen oder Volumen, klassifiziert nach ihren Dicken. Für die Berechnung muß demnach die Länge und die Höhe einer Wand ermittelt werden können. Die VOB-C definiert Regeln für die Berechnung dieser Werte, wenn Wände beispielsweise mit anderen Wänden oder mit Decken verbunden werden. So ergibt sich die Höhe einer Wand, wenn sie sich unter einer Decke befindet, aus ihrer tatsächlichen Höhe. Ansonsten berechnet sich die Höhe bis zur Oberkante der Decke. Ein weiterer Spezialfall ergibt sich, wenn sich Wände kreuzen. Hierzu wird die dickere Wand durchgemessen. Abbildung 5.2 stellt die Spezialfälle noch einmal grafisch dar.

Des weiteren muß die Mengenermittlung Abzugsflächen berücksichtigen. Laut VOB-Teil C müssen Öffnungen bei der Mengenberechnung nach dem Flächenmaß beachtet werden, deren Oberfläche größer als 1,0 m^2 sind. Wird die Berechnung nach Raummaß durchgeführt, so müssen Öffnungen, die über 0.25 m^3 sind, vom Volumen der Wand abgezogen werden. Alle weiteren

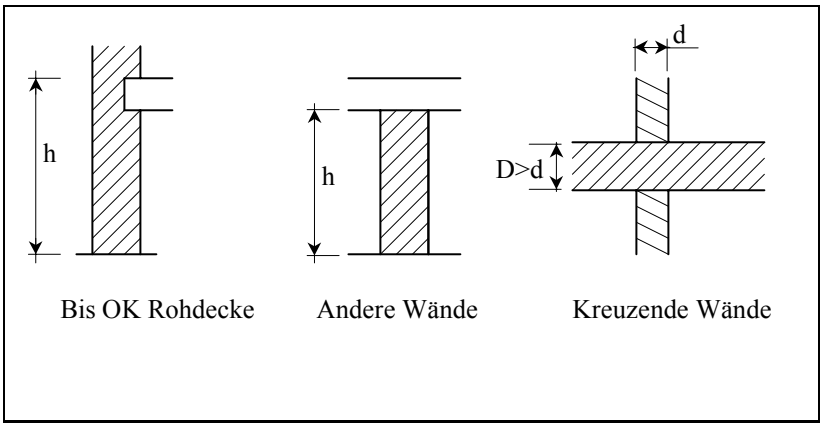


Abbildung 5.2: Regeln für die Mengenermittlung einer Wand (Quelle [Mant91])

Regeln der VOB- Teil C können an dieser Stelle unberücksicht bleiben, da diese Regeln im Präsentationsbeispiel nicht anzutreffen sind.

Die Mengenermittlung der Unterzüge, Fenster und Türen soll sich aus der Anzahl von gleichartigen Bauteilen ergeben, wie diese oft im Grundriß anzutreffen sind. Demnach müssen Positionen für alle Stürze sowie für die Fenster und Türen erzeugt werden. Der Mengenansatz für die Position der Stürze muß für jede unterschiedliche Länge einen entsprechenden Eintrag beinhalten. Die Mengenansätze der Positionen Fenster und Türen listen verschiedenartige Objekte auf.

5.2 Übernahme der Mengendaten

In den Abschnitten 3.2.2 und 4.2.3 (siehe Seite 40 und 85) wurde der prinzipielle Ablauf der Datenübernahme dargestellt. Demnach beruht dieser Prozeß auf folgenden Teilaufgaben:

- Selektion der Objekte in dem Modell, welches Informationen für die Übernahme bereitstellt und für die ein und derselbe Abbildungsweg existiert.
- Erzeugen der Typstruktur sowie des Objektzustandes der übernommenen Objekte im persistenten und temporären Datenraum des Integrationsmoduls, wenn noch nicht vorhanden.
- Manipulation der selektierten Objekte.
- Erzeugen der Objekte in dem Modell, welches die manipulierten Daten übernehmen soll.
- Erzeugen der Typstruktur und des Objektzustandes der neu eingefügten Objekte im persistenten Datenraum des Integrationsmoduls.
- Generierung der Abhängigkeiten zwischen den selektierten Objekten und den neu eingefügten Objekten.

Entsprechend den Regeln der VOB müssen für die Übernahme der Mengen verschiedene Auswahlsätze erzeugt werden. Für die Übernahme der Wände werden drei Auswahlsätze benötigt.

Die Übernahme aller Türen und Fenster erzeugt zwei neue Auswahlsätze. Der Grund hierfür ist, daß die Wände diese Objekte aggregieren und daß laut Voraussetzung nur Objekte des gleichen Typs Elemente eines Auswahlsatzes sein dürfen. Für die Übernahme der Unterzüge und für die Übernahme der Decke wird jeweils ein weiterer Auswahlsatz benötigt. Die folgende Aufzählung beschreibt die einzelnen Auswahlsätze, die für die Mengenermittlung notwendig sind:

- alle Außenwände mit der Dicke von 0.36 m ⇒ (1),
- alle Innenwände mit der Dicke von 0.36 m ⇒ (2),
- alle Innenwände mit der Dicke von 0.24 m ⇒ (3),
- Fenster ⇒ (4),
- Türen ⇒ (5),
- Unterzüge ⇒ (6),
- Decken ⇒ (7).

Die Abbildung 5.3 zeigt die Zuordnung der einzelnen Bauelemente zu den einzelnen Auswahlsätzen sowie deren Identifikation im Integrationsmodul. Die Abbildung 5.4 zeigt exemplarisch die in das Integrationsmodul übernommene Objektbeschreibung sowie den Zustand der längsten Außenwand des CAD-Modells.

Für die Mengenermittlung müssen die Prinzipien des Modellierungskonzeptes vom Architekten beachtet werden. Die wesentliche Aufgabe des Architekten ist es, einen architektonischen Entwurf anzufertigen, um das Aussehen des Bauwerkes anzugeben. Aus diesem Grund beinhaltet das CAD-Modell nicht die korrekte Anzahl sowie die korrekten Mengen der Bauteile, sondern gibt nur die resultierende Gestalt des Bauwerkes wieder. Beispielsweise modelliert der Architekt weder Stürze über den Fenstern oder Türen, noch beachtet er unterschiedliche Angaben für die Höhe der einzelnen Wände, die für die Mengenermittlung benötigt werden.

5.2.1 Mengenermittlung der Wände

Die Aufgabe der nun folgenden Kommandos ist es, die inadäquate Modellierung des CAD-Modells für die Aufgaben der Mengenermittlung durch das gezielte Auswerten von übernommenen Daten zu beseitigen, um die korrekten Mengen entsprechend der VOB-C zu ermitteln. Demnach werden für die Mengenermittlung der Wände folgende Schritte notwendig:

1. Erweiterung des Wand-Objekttyps auf die benötigten Attribute des Mengenansatzes,
2. Einfügen der Wandmengen in die neu bereitgestellte Struktur,
3. Erzeugen der Wandabzugsvolumen durch die Manipulation der referenzierten Fenster- und Türobjekte und
4. Erstellen der einzelnen AVA-Wandpositionen.

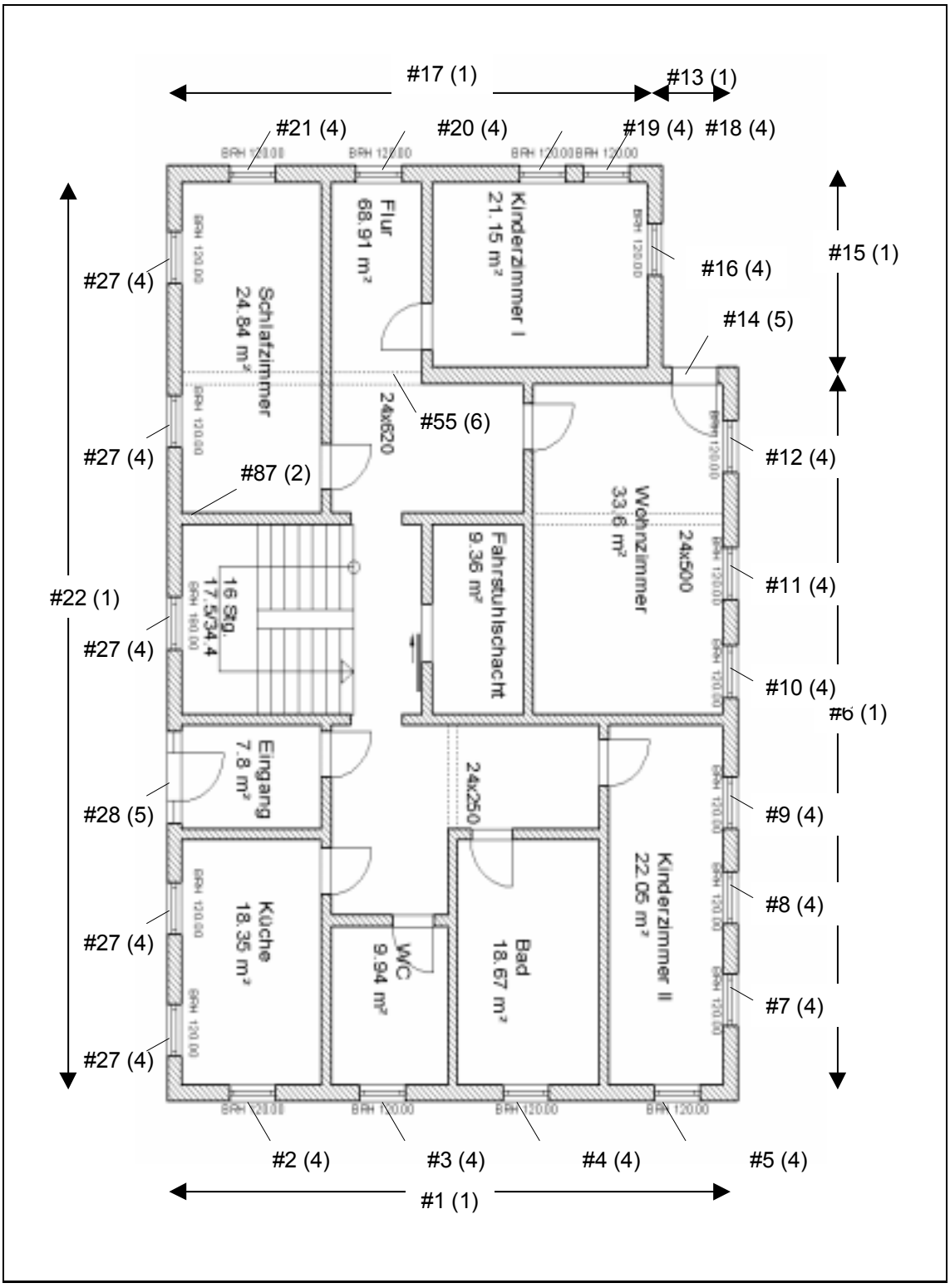


Abbildung 5.3: Zuordnung der Elemente zu den einzelnen Auswahlsätzen

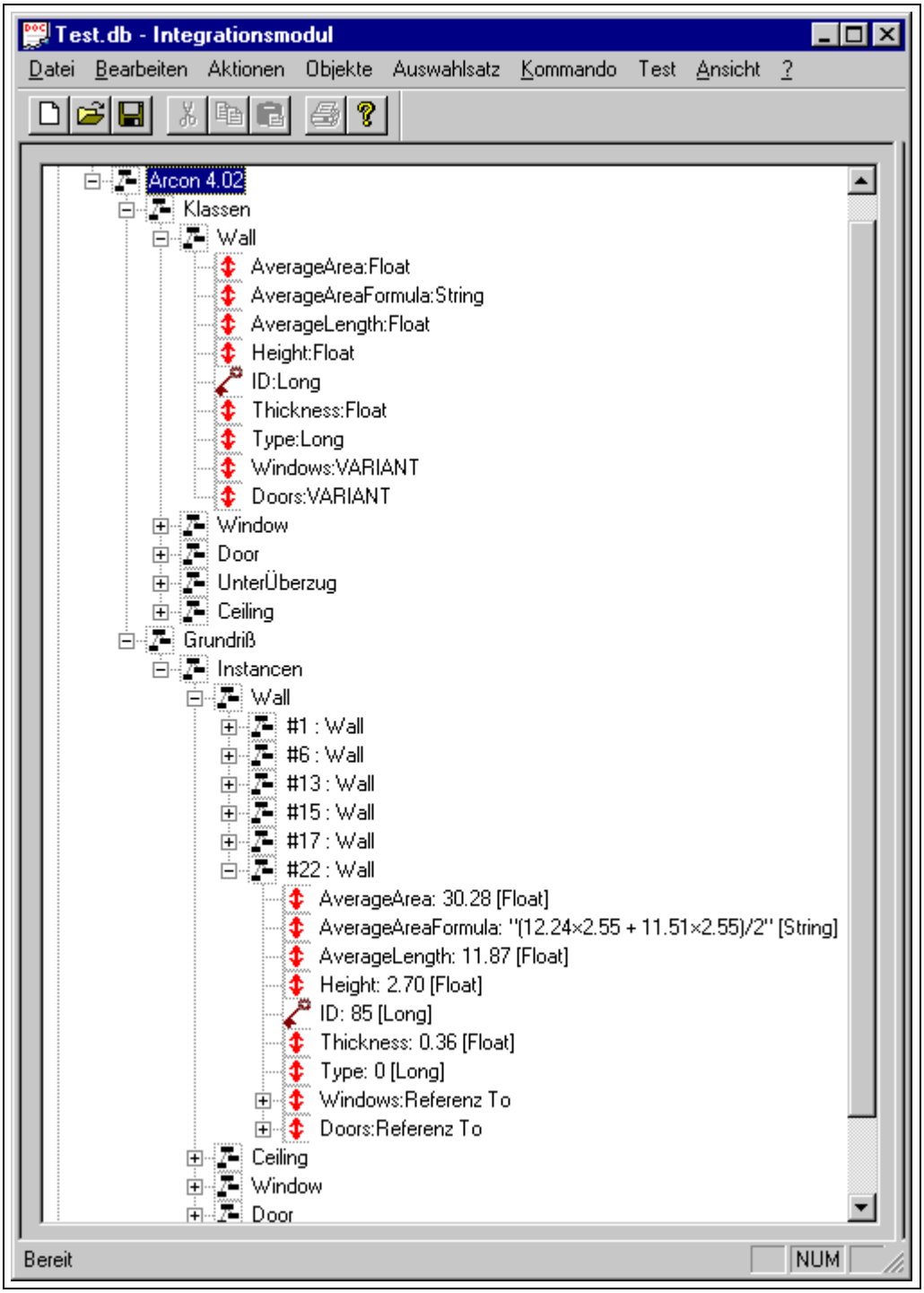


Abbildung 5.4: Objekttyp der südlichen Außenwand

Die folgende Auflistung gibt einen Überblick über die verwendeten Kommandos. Hierzu wird als erstes eine kurze Erläuterung des Inhaltes des Kommandos angegeben. Die nächste Zeile beinhaltet den Namen des resultierenden Auswahlsatzes.

- Erweitern des Datentyps um die Attribute des Mengenansatzes.
  Name: CAD_Wand
- Einfügen des Wandvolumens als ersten Eintrag der Mengenermittlung in die gerade geschaffene Datenstruktur.
  Name: UpdateWandmenge
- Umwandlung der Referenztypen *Windows* und *Doors* in Mengentypen.
  Name: UpdateReferenztypen
- Projektion aller Attribute auf die für die Mengenermittlung benötigten Attribute.
  Name: Projektion_AVAWand
- Erweitern der Attribute *Windows* und *Doors* um die Attribute der Mengenermittlung und um das Attribut der Wanddicke.
  Name: MengenAttributeOeffnungen
- Übernahme der Dicken der Wände in die Attribute *Windows* und *Doors*.
  Name: UpdateDicke
- Auswahl aller Öffnungen, deren Volumen > 0.25 m^3 sind.
  Name: SelectOeffnungen
- Aktualisieren der Attribute des Mengenansatzes.
  Name: UpdateOeffnungen
- Aufzählen gleicher Öffnungsmaße.
  Name: CountGleicheÖffnungsmaße
- Addieren der Instanzen von Window und Doors zu MAnsatz.
  Name: AVA_MAnsatz
- Projizieren aller Wände nur noch auf ID und MAnsatz.
  Name: ProjectionMAnsatz
- Erweitern des Datentyps um Erläuterung.
  Name: ExtendErläuterung
- Aktualisieren des Attributes Erläuterung mit den Werten Wand, ID und Dicke.
  Name: AVA_Position

Im folgenden werden nun die notwendigen Abfragen angegeben. Hierzu wird als erstes das Kommando angegeben, danach der neue Auswahlsatzname. Für einige Operationen ist es für das Verständnis des Kommandos zweckmäßig, die resultierende Datenstruktur mit einem Exemplar des Auswahlsatzes anzugeben.

Das erste Kommando erweitert den Objekttyp aller im Auswahlsatz *Außenwand_36* enthaltenen Objekte.

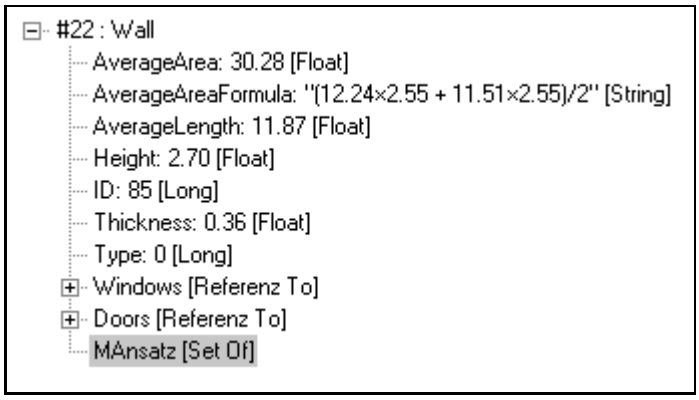


Abbildung 5.5: Darstellung des Präsentationsobjektes im Auswahlsatz CAD_Wand

Kommando:
*Außenwand_36.*EXTEND (MAnsatz: SET OF [TUPEL OF < Erläuterung : STRING,
Vorzeichen : STRING,
Faktor : FLOAT,
Ansatz : STRING >]) ;

Name: CAD_Wand

Abbildung 5.5 stellt die Attributisierung der längsten Wand des Modells in der Notation eines Baumes dar. Die dafür benutzte Datenstruktur besitzt folgenden Aufbau:

$CAD\_Wand = \mathcal{K}_T(\{$<AverageArea,Float,normal> ,
<AverageAreaFormula,String,normal> ,
<AverageLength,Float,normal> ,
<Height,Float,normal> ,
<ID,Long,key> ,
<Thickness,Float,normal> ,
<Type,Long,normal> ,
<Windows, $\mathcal{K}_{Ref}$,normal> ,
<Doors, $\mathcal{K}_{Ref}$,normal> ,
<MAnsatz, $\mathcal{K}_S$ [$\mathcal{K}_T(\{$ <Erläuterung, String, normal>,
<Vorzeichen, String, normal>,
<Faktor, Float, normal>,
<Ansatz, String, normal>
$\})$
],normal> ,
$\})$.

Nachdem der Datentyp der ARCON-Wand um den Datentyp des Mengenansatzes erweitert wurde, kann nun die Menge für jede Wand als erster Datensatz in die erweiterte Struktur eingefügt werden.

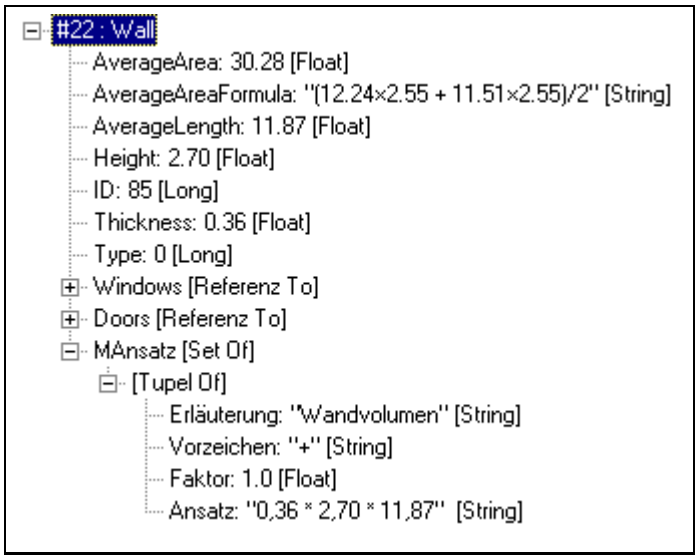


Abbildung 5.6: Darstellung des Objektes im Auswahlsatz UpdateWandmenge

Kommando:
CAD_ Wand. (UPDATE [MAnsatz] USES [ID], [Thickness], [Height], [AverageLength] SET
[MAnsatz] += < [Erläuterung] = 'Wandvolumen',
[Vorzeichen] = '+',
[Faktor] = 1.0 ,
[Ansatz] = " + [Thickness] + '*' + [Height] + '*' + [AverageLength] >) ;

Name: UpdateWandmenge

Die Formulierung des Kommandos setzt voraus, daß die exakten Mengen der einzelnen Wände ermittelt werden sollen. Denkbar ist es des weiteren durch die Manipulation der Größe *Faktor* Zulagen oder Abminderungen anzugeben. Abbildung 5.6 zeigt noch einmal das entstehende Objekt der längsten Wand, nachdem das Kommando auf das Objekt angewendet wurde.

Das nächste Kommando konvertiert die beiden Attribute *Windows* und *Doors* in einen Mengentyp um. Hintergrund für dieses Kommando ist es, die referenzierten Datentypen von *Windows* und *Doors* manipulieren zu können. Das Kommando lautet:

Kommando:
UpdateWandmenge.UPDATE [Windows], [Doors]
SET [Windows].CREATESET(), [Doors].CREATESET().

Name: UpdateReferenztypen

Die folgende Typbeschreibung zeigt den Aufbau der im Auswahlsatz *UpdateReferenztypen* enthaltenen Objekte.
UpdateReferenztypen = $\mathcal{K}_T$({<AverageArea,Float,normal> ,
<AverageAreaFormula,String,normal> ,
<AverageLength,Float,normal> ,
<Height,Float,normal> ,

<ID,Long,key> ,
<Thickness,Float,normal> ,
<Type,Long,normal> ,
<Windows, $\mathcal{K}_S$[$\mathcal{K}_T$({<Area,Float,normal> ,
<AreaFormula,String,normal> ,
<Height,Float,normal> ,
<ID,Long,key> ,
<LeftHung,Bool,normal> ,
<OpensInwards,Bool,normal> ,
<ParapetHeight,Float,normal> ,
<Remark,String,normal> ,
<Type,Long,normal> ,
<Width,Float,normal>
})
], normal>,
<Doors, $\mathcal{K}_S$[$\mathcal{K}_T$({<Area,Float,normal> ,
<AreaFormula,String,normal> ,
<Height,Float,normal> ,
<HungLeft,Bool,normal> ,
<ID,Long,key> ,
<LeftWingAngle,Float,normal> ,
<LeftWingType,Long,normal> ,
<OpensInwards,Bool,normal> ,
<Remark,String,normal> ,
<RightWingAngle,Float,normal> ,
<RightWingType,Long,normal> ,
<Type,Long,normal> ,
<Width,Float,normal>
})
], normal>,
<MAnsatz, $\mathcal{K}_S$[$\mathcal{K}_T$({<Erläuterung, String, normal>,
<Vorzeichen, String, normal>,
<Faktor, Float, normal>,
<Ansatz, String, normal>
})
],normal>
})

Bevor die Attributisierung des Beispielobjektes angegeben wird, soll noch ein weiteres Kommando beschrieben werden. Das Kommando projiziert alle Attribute auf die gewünschten Attribute. Hierzu wird ein neuer Datentyp angegeben, der nur noch die gewünschten Attribute beinhaltet. Das Kommando besitzt folgendes Aussehen:

Kommando:
UpdateReferenztypen.PROJECTION
(ID : LONG ,
Thickness : FLOAT,
Windows : SET OF [TUPEL OF < Height : FLOAT, ID : LONG, Width : FLOAT >] ,
Doors : SET OF [TUPEL OF < Height : FLOAT, ID : LONG, Width : FLOAT >] ,

```
    MAnsatz : SET OF [ TUPEL OF < Erläuterung : STRING,
                                  Vorzeichen : STRING,
                                  Faktor : FLOAT,
                                  Ansatz : STRING
                                >
                     ]
) ;
```

Name: Projektion_AVAWand

Nach dem Kommando beinhalten die Objekte des Auswahlsatzes *AVA_ProjNurWand* folgenden Objekttyp:

Projektion_AVAWand = $\mathcal{K}_T$({<ID,Long,key> ,
<Thickness,Float,normal> ,
<Windows, $\mathcal{K}_S$($\mathcal{K}_T$({<Height,Float,normal> ,
<ID,Long,key> ,
<Width,Float,normal>
})), normal>,
<Doors, $\mathcal{K}_S$($\mathcal{K}_T$({<Height,Float,normal> ,
<ID,Long,key> ,
<Width,Float,normal>
})) , normal> ,
<MAnsatz, $\mathcal{K}_S$($\mathcal{K}_T$({<Erläuterung, String, normal>,
<Vorzeichen, String, normal>,
<Faktor, Float, normal>,
<Ansatz, String, normal>,
})),normal>
})

Abbildung 5.7 zeigt die Attributisierung des Präsentationsexemplars in seiner Darstellung als Baum.

Mit den folgenden Anweisungen werden die Datentypen der Attribute *Windows* und *Doors* um die Attribute des Mengenansatzes sowie um ein neues Attribut für die einzelnen Wanddicken erweitert und anschließend die entsprechenden Wanddicken in den erweiterten Mengentypen übernommen.

Kommando:

```
Projektion_AVAWand.EXTEND (
    Windows: SET OF [ TUPEL OF < Thickness : FLOAT,
                                 Erläuterung : STRING,
                                 Vorzeichen : STRING,
                                 Ansatz : STRING >
                    ] ,
    Doors: SET OF [ TUPEL OF < Thickness : FLOAT,
                               Erläuterung : STRING,
                               Vorzeichen : STRING,
                               Ansatz : STRING >
                  ] ) ;
```

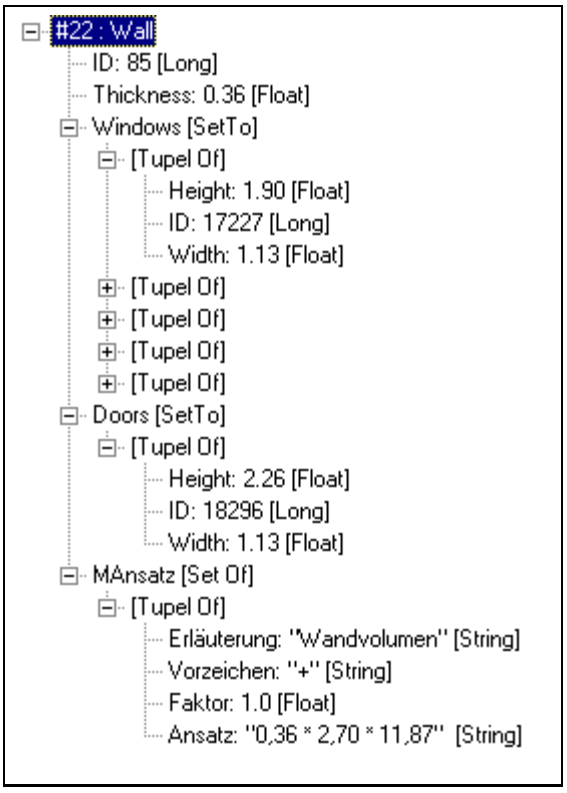


Abbildung 5.7: Darstellung eines Objektes im Auswahlsatz Projektion_AVAWand

Name: MengenAttributeOeffnungen

Kommando:
MengenAttributeOeffnungen.UPDATE ([Windows],[Doors] USES [Thickness] SET
Windows.Update(<Thickness = [Thickness]>),
Doors.Update(<[Thickness] = [Thickness]>)
) ;

Name: UpdateDicke

Im Anschluß daran müssen die Objekte ausgewählt werden, deren Öffnungsvolumen größer als 0.25 m^3 sind. Dafür wird die Methode SELECT der einzelnen Mengentypen aufgerufen. Das entsprechende Kommando lautet:

Kommando:
UpdateDicke.UPDATE ([Windows],[Doors] SET
[Windows].SELECT(([Thickness] * [Height] * [Width]) > 0.25),
[Doors].SELECT(([Thickness] * [Height] * [Width]) > 0.25) ;

Name: SelectOeffnungen

Die folgenden Kommandos realisieren die Aktualisierung der Attribute des Mengenansatzes in den Wandattributen *Windows* und *Doors*. Hierzu werden als erstes die Attribute des Mengen-

ansatzes aktualisiert und danach die Aggregatfunktion COUNT für die Bestimmung gleichartiger Öffnungen angewendet.

Kommando:
SelectOeffnungen.UPDATE ([Windows], [Doors] SET
[Windows].UPDATE (USES [Thickness], [Height], [Width]
< [Erläuterung]= 'Fenster',
[Vorzeichen] = '-',
[Ansatz] = '' + [Thickness] + '*' + [Height] + '*' + [Width]
>),
[Doors].UPDATE (USES [Thickness], [Height], [Width]
< [Erläuterung] = 'Türen',
[Vorzeichen] = '-',
[Ansatz] = '' + [Thickness] + '*' + [Height] + '*' + [Width]
>)) ;

Name: UpdateOeffnungen

Kommando:
UpdateOeffnungen.UPDATE ([Windows], [Doors] SET
[Windows].COUNT(TUPEL OF < Erläuterung : STRING,
Vorzeichen : STRING,
Ansatz : STRING > , Faktor : FLOAT),
[Doors].COUNT(TUPEL OF < Erläuterung : STRING,
Vorzeichen : STRING,
Ansatz : STRING > , Faktor : FLOAT)) ;

Name: CountGleicheÖffnungsmaße

Die Abbildung 5.8 zeigt die Attributisierung des Präsentationsobjektes. Die resultierende Datenstruktur besitzt folgenden Aufbau:

CountGleicheÖffnungsmaße = $\mathcal{K}_T$({<ID,Long,key> ,
<Thickness,Float,normal> ,
<Windows, $\mathcal{K}_S$ ($\mathcal{K}_T$({ <Faktor, Float,normal>,
<Erläuterung, String,normal>,
<Vorzeichen, String,normal>,
<Ansatz, String,normal>
})), normal >,
<Doors, $\mathcal{K}_S$ ($\mathcal{K}_T$({ <Faktor, Float,normal>,
<Erläuterung, String,normal>,
<Vorzeichen, String,normal>,
<Ansatz, String,normal>
})), normal >,
<MAnsatz, $\mathcal{K}_S$($\mathcal{K}_T$ ({ <Erläuterung, String, normal>,
<Vorzeichen, String, normal>,
<Faktor, Float, normal>,
<Ansatz, String, normal>,
})),normal>
})

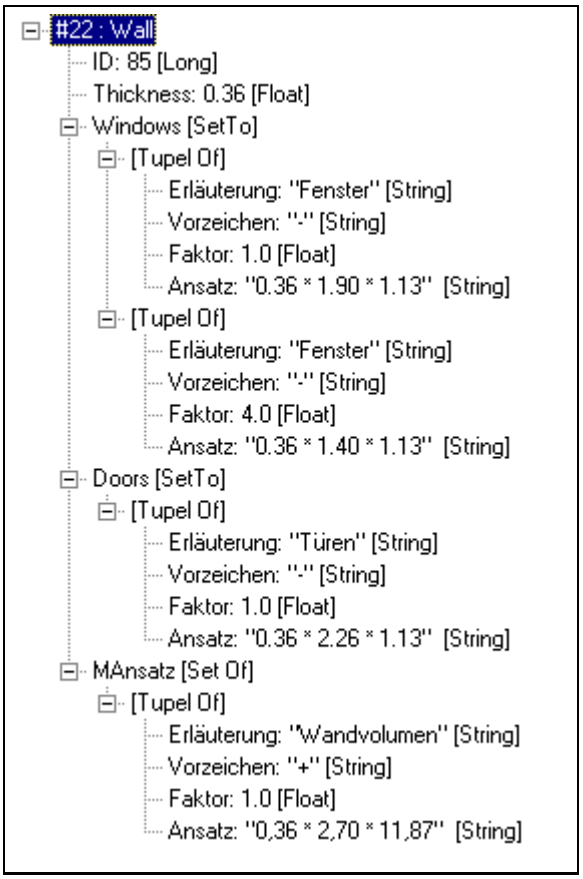


Abbildung 5.8: Präsentationsobjekt in *CountGleicheÖffnungsmaße*

Die so erhaltenen Datensätze können anschließend mittels einer UPDATE-Anweisung in dem Attribut *MAnsatz* zusammengefaßt werden.

Kommando:
CountGleicheÖffnungsmaße.UPDATE ([MAnsatz] USES [Windows],[Doors]
SET
[MAnsatz] += [Windows] + [Doors]
) ;

Name: AVA_MAnsatz

Das nächste Kommando projiziert den Wanddatentyp des Auswahlsatzes *AVA_MAnsatz* auf den gewünschten, für die Aufgaben der Mengenermittlung zugeschnittenen Objekttyp, d.h. die Attribute *Windows* und *Doors* werden aus der Objektbeschreibung entfernt. Anschließend müssen noch die notwendigen Attribute einer AVA-Position in den veränderten Objekttyp eingefügt und mit sinnvollen Attributwerten belegt werden. Die fehlenden Kommandos lauten:

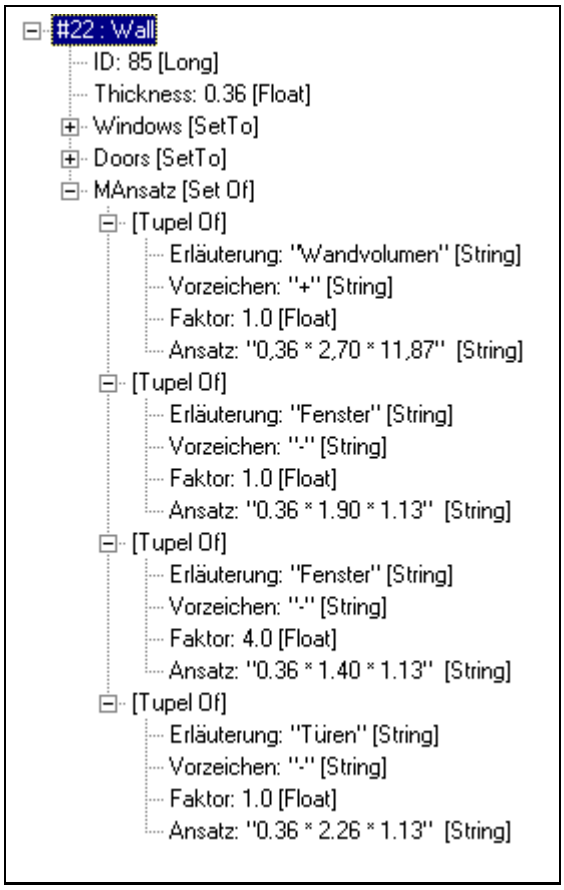


Abbildung 5.9: Präsentationsobjekt in *AVA_MAnsatz*

Kommando:
AVA_MAnsatz.Projection (MAnsatz : Set Of [Tupel Of < Erläuterung : STRING,
Vorzeichen : STRING,
Faktor : FLOAT,
Ansatz : STRING >] ,
ID : Long,
Thickness : Float
) ;

Name: ProjectionMAnsatz

Kommando:
ProjectionMAnsatz.EXTEND (Erläuterung : STRING) ;

Name: ExtendErläuterung

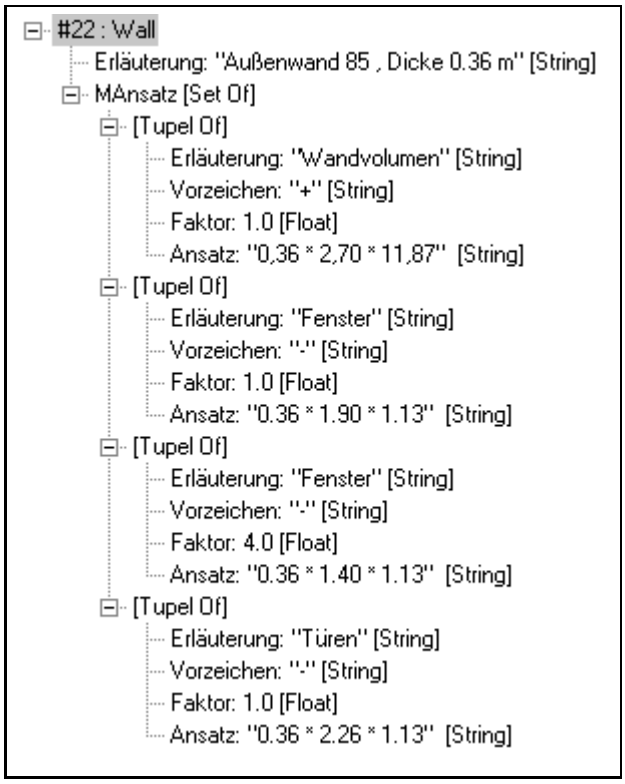


Abbildung 5.10: Präsentationsobjekt in *AVA_Position*

Kommando:
ExtendErläuterung.UPDATE ([Erläuterung] USES [ID], [Thickness]
SET
[Erläuterung] = 'Außenwand' + [ID] +' , Dicke = ' + [Thickness] + 'm'
) ;

Name: AVA_Position

Die resultierende Datenstruktur entspricht weitestgehend der unter 5.1.2 angegebenen Datenstruktur und besitzt folgenden Aufbau:

$$
\begin{aligned}
AVA\_Position = \mathcal{K}_T(\{&<\text{Erläuterung,String,normal}> \ , \\
&<\text{MAnsatz}, \mathcal{K}_S(\mathcal{K}_T \ (\{ <\text{Erläuterung, String, normal}>, \\
&\qquad <\text{Vorzeichen, String, normal}>, \\
&\qquad <\text{Faktor, Float, normal}>, \\
&\qquad <\text{Ansatz, String, normal}>, \\
&\qquad \}) \ ),\text{normal}> \\
&\})
\end{aligned}
$$

Abbildung 5.10 stellt letztendlich die längste Außenwand des Grundrisses nach der Manipulation dar. Die Übernahme aller Objekte des Auswahlsatzes *AVA_Position* ermöglicht eine nachvollziehbare Mengenermittlung in einem Dokument des AVA-Systems. Bevor jedoch die Übernahme gestartet werden kann, muß ein Dokument des AVA-Systems in das Integrationsmodul eingebettet werden.

Abbildung 5.11: Dialogbox, Übernahme der Objekte

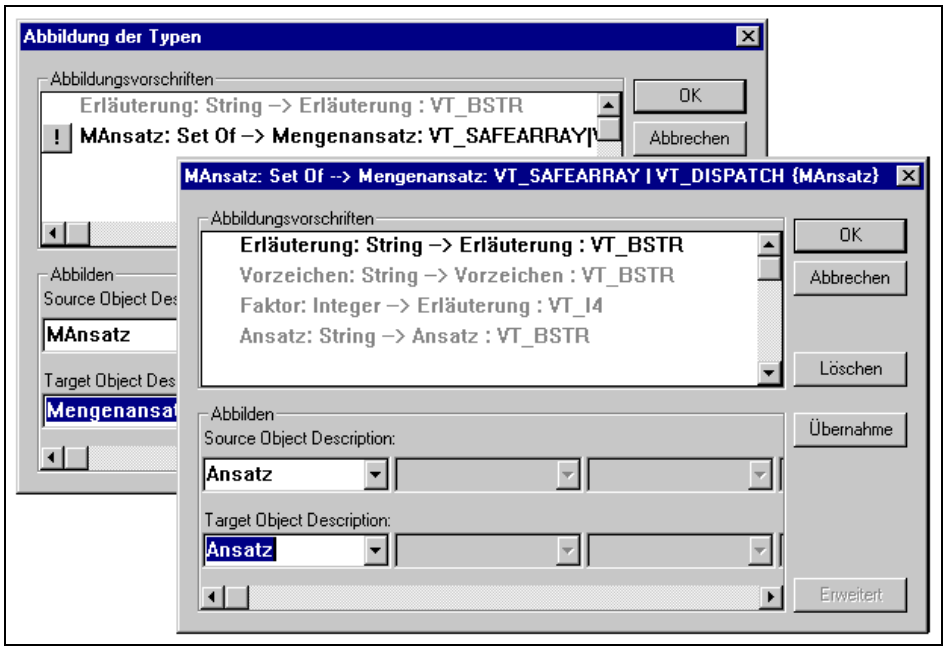


Abbildung 5.12: Dialogbox, Abbildung der Typen

Die Übernahme der Wandmengen beginnt mit einer Dialogbox, in der der Auswahlsatz, welcher die zu übernehmenden Objekte beinhaltet, das Zielmodell sowie die unterstützte Objektbeschreibung des Zielmodells ausgewählt werden müssen. Abbildung 5.11 zeigt die Dialogbox mit den entsprechenden anzugebenden Werten.

Im Anschluß daran müssen noch die einzelnen Abbildungsvorschriften zwischen dem Typsystem des Integrationsmoduls und dem Typsystem der Automation-Schnittstelle angegeben werden. Für dieses Fallbeispiel muß der Aufbau des VARIANT-Typs von MAnsatz näher erläutert werden. Abbildung 5.12 zeigt die notwendigen Eingaben, die noch für die Übernahme angegeben werden müssen.

Das Akzeptieren der Eingaben führt zur Übernahme des Mengenansatzes. Hierbei wird für jedes Objekt des Auswahlsatzes *AVA_Position* ein neues Objekt in dem Dokument des AVA-Systems angelegt. Die Übernahme der Objekte erfolgt schrittweise. Demnach wird zunächst das erste Objekt im AVA-System betrachtet. Nachdem von den angegebenen Attributen die Werte übergeben worden sind, wird das AVA-Objekt in den persistenten Datenraum des Integrationsmoduls

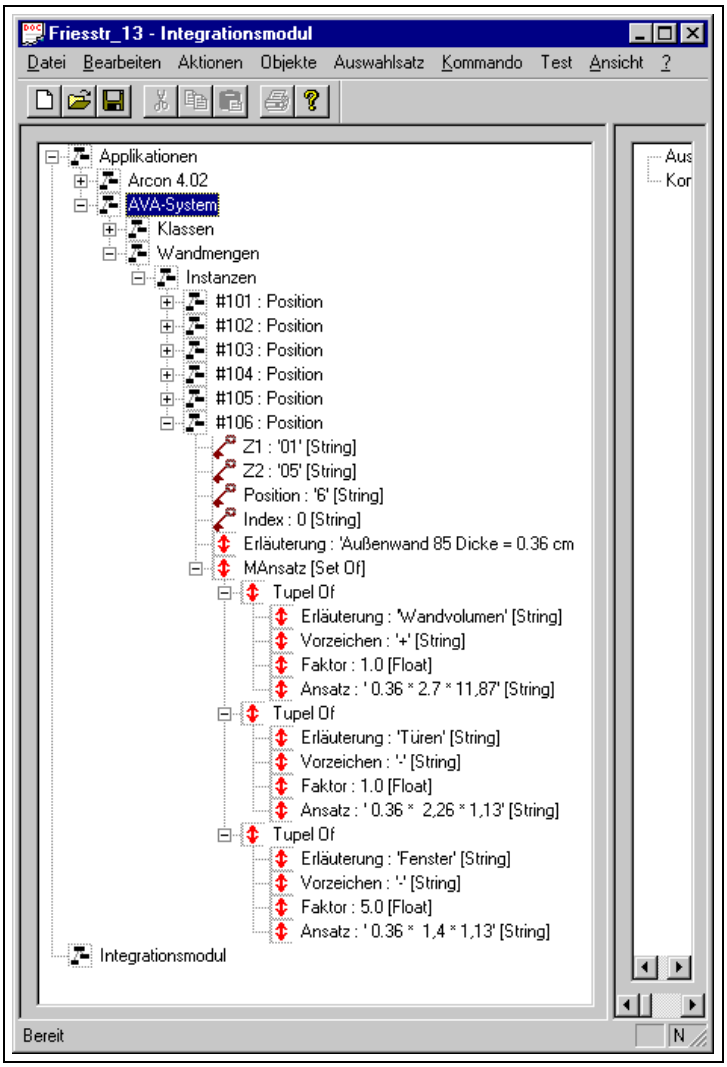


Abbildung 5.13: Persistente Objekte des AVA-Modelles *Wandmengen* im Integrationsmodul

übernommen und die Abhängigkeiten in die Relation CIMO_DEPENDENCE eingetragen. Danach wird das nächste Objekt betrachtet, bis alle Objekte des Auswahlsatzes übernommen worden sind.

Die Übernahme des Attributes MAnsatz führt zur Erzeugung von neuen Instanzen basierend auf dem gleichnamigen Datentyp im AVA-System. Das Integrationsmodul sammelt von diesen erzeugten Instanzen deren Dispatchzeiger in einem Array und übergibt dieses Array dem Attribut *Mengenansatz* der AVA-Position, nachdem alle Elemente vom Attribut MAnsatz im AVA-System erzeugt wurden.

Die Abbildung 5.13 zeigt die resultierenden Datenstrukturen des AVA-Systems, die sich im persistenten Datenraum des Integrationsmoduls befinden. Die neu eingefügten Relationen zeigt Tabelle 5.3.

Wandreferenz	Fensterreferenzen	Türreferenzen	Position
#1	#2, #3, #4, #5		#101
#6	#7, #8, #9, #10, #11, #12		#102
#13	#14		#103
#15		#16	#104
#17	#18, #19, #20, #21		#105
#22	#23, #24, #25, #26, #27	#28	#106

Tabelle 5.3: Generierte Relationen durch die Übernahme der Wandmengen

Die Übernahme der Mengen für die Innenwände des CAD-Plans beruht auf dem gleichen Prinzip wie die der Außenwände. Verändert werden müssen jedoch folgende Kommandos:

- CAD_Wand: Zuweisen eines anderen zugrundeliegenden Auswahlsatzes,
- ExtendErläuterung: Verändern des Attributwertes *Erläuterung* in Innenwand,
- Definition neuer Kommandos, um die Höhe einer Wand um die Dicke der Decke zu vermindern.

Die folgenden Kommandos zeigen den Weg, wie die Höhe einer Wand um die Dicke der Decke vermindert werden kann. Hierzu muß als erstes das Attribut *Thickness* des Datentyps *Ceiling* umbenannt werden, und danach müssen alle Wandobjekte mit dem Deckenobjekt kombiniert werden. Im Anschluß daran wird mittels einem UPDATE das Attribut *Height* manipuliert. Die Kommandos lauten:

Kommando:
Ceiling.RENAME ([Thickness] ALIAS DeckenDicke) ;

Name: RenameThickness

Kommando:
Innenwand_36.JOIN (*RenameThickness*) ;
Innenwand_24.JOIN (*RenameThickness*) ;

Name: JoinWandMitDecke

Kommando:
JoinWandMitDecke.UPDATE ([Height] USES [DeckenDicke] SET [Height] -= [DeckenDicke]);

Name: AllgemeineWand;

5.2.2 Mengenermittlung der Fenster und der Türen

Die Abfragen der Mengenermittlung von den Fenstern und den Türen haben zum Ziel, die Anzahl gleichartiger Objekte zu ermitteln. Für jede unterschiedliche Tür oder für jedes unterschiedliche Fenster wird ein neuer Mengenansatz definiert. Zwei neue Positionen, eine Position für die Mengen der Fenster und eine weitere für die Menge der Türen, müssen ebenfalls angelegt werden.

Das prinzipielle Vorgehen setzt sich aus folgenden Arbeitsschritten zusammen:

- Generieren eines neuen Auswahlsatzes: Der Objekttyp setzt sich aus den beiden Attributen *Erläuterung* und *MAnsatz* zusammen, wobei *Erläuterung* vom Typ *String* und *MAnsatz* vom Typ *Reference To* ist.
  Name: CreateAVA_Position
- Generieren eines neuen Auswahlsatzes, der auf dem Auswahlsatz *Windows* basiert, jedoch nur noch eine Teilmenge aller Attribute beinhaltet. Die Teilmenge setzt sich aus den Attributen zusammen, die die benötigten Attribute der Mengenermittlung darstellen.
  Name: ProjektionFenster
- Generieren eines neuen Auswahlsatz, der die gleichen Attribute wie der Auswahlsatz *ProjektionFenster* beinhaltet, der jedoch auf den Objekten des Auswahlsatzes *Doors* basiert.
  Name: ProjektionTüren
- Erweiterung des Datentyps der Objekte der Auswahlsätze *ProjektionFenster* und *ProjektionTür* um die Attribute des vom AVA-Systems verwendeten Datentyps MAnsatz.
  Namen: AVA_Fenster und AVA_Tür
- Aktualisierung der Attribute des Datentyps MAnsatz in den Auswahlsätzen *AVA_Fenster* und *AVA_Tür*.
  Namen: UpdateAVA_Fenster und UpdateAVA_Tür
- Einfügen eines neuen Objektes in den Auswahlsatz *CreateAVA_Position*, wobei das Attribut *Erläuterung* mit der Zeichenkette 'Mengenermittlung Fenster' initialisiert und das Attribut *MAnsatz* mit den Objekten des Auswahlsatzes *UpdateAVA_Fenster* verbunden wird.
  Name: InsertFenster
- Einfügen eines neuen Objektes in den Auswahlsatz *CreateAVA_Position*, wobei das Attribut *Erläuterung* mit der Zeichenkette 'Mengenermittlung Türen' initialisiert wird und das Attribut *MAnsatz* mit den Objekten des Auswahlsatzes *UpdateAVA_Tür* verbunden wird.
  Name: InsertFensterUndTüren
- Umwandeln des Datentyps des Attributes *MAnsatz* in einen Mengendatentyp.
  Name: UpdateReferenz
- Ermittlung der Kardinalitäten gleichartiger Objekte.
  Name: AVA_Position

Das Beispiel soll an dieser Stelle beendet werden, obwohl noch nicht alle notwendigen Kommandos angegeben wurden. Die fehlenden Kommandos müssen das Attribut *Type* noch auswerten, um

die Konstruktion der Fenster- und Türobjekte genauer zu beschreiben. Beispielsweise beschreibt ein Fenstertyp mit dem Wert 2 ein Fenster, welches aus zwei Flügeln besteht, die doppelt verglast sind. Eine Weiterführung des Beispiels würde eine detaillierte Beschreibung der Objekttypen des ARCON CAD-Systems nach sich ziehen, worauf in dieser Arbeit verzichtet werden soll.

Die folgenden Kommandos realisieren die Abbildung der Fenster- und Türobjekte in die Datenstruktur des AVA-Systems:

Kommando: **Name:** CreateAVA_Position

```
CREATE ( Erläuterung: STRING, MAnsatz: REFERENCE TO () ) ;
```

Kommando: **Name:** ProjektionFenster

```
Windows.PROJECTION ( ID : LONG ,
                     !Type : LONG,
                     Height : FLOAT,
                     Width : FLOAT
                   ) ;
```

Kommando: **Name:** ProjektionTüren

```
Doors.PROJECTION ( ID : LONG ,
                   !Type : LONG,
                   Height : FLOAT,
                   Width : FLOAT
                 ) ;
```

Kommando: **Name:** AVA_Fenster

```
ProjektionFenster.EXTEND ( Erläuterung : STRING,
                           Vorzeichen : STRING,
                           Ansatz : STRING
                         ) ;
```

Kommando: **Name:** AVA_Türen

```
ProjektionTüren.EXTEND ( Erläuterung : STRING,
                         Vorzeichen : STRING,
                         Ansatz : STRING
                       ) ;
```

Kommando: **Name:** UpdateAVA_Fenster

```
AVA_Fenster.UPDATE ( [Erläuterung] USES [!Type], [Width], [Height]
SET
   [Erläuterung] = 'Fenster, Breite = ' + [Width] +
                   ', Höhe = ' + [Height] +
                   ', und Typ = ' + [!Type]
) ;
```

Kommando: **Name:** UpdateAVA_Türen

AVA_Türen.UPDATE ([Erläuterung] USES [!Type], [Width], [Height]
SET
[Erläuterung] = 'Türen, Breite = ' + [Width] +
', Höhe = ' + [Height] +
', und Typ = ' + [!Type]
) ;

Kommando: **Name:** InsertFenster

CreateAVA_Position.INSERT (
UPDATE [Erläuterung], [MAnsatz]
SET [Erläuterung] = 'Mengenermittlung Fenster',
[MAnsatz].SET (*UpdateAVA_Fenster*)

) ;

Kommando: **Name:** InsertFensterUndTüren

InsertFenster.INSERT (
UPDATE [Erläuterung], [MAnsatz]
SET [Erläuterung] = 'Mengenermittlung Türen',
[MAnsatz].SET (*UpdateAVA_Türen*)

) ;

Kommando: **Name:** UpdateReferenz

InsertFensterUndTüren.UPDATE ([MAnsatz] SET [MAnsatz].CREATESET()) ;

Kommando: **Name:** AVA_Position

UpdateReferenz.UPDATE [MAnsatz] SET
[MAnsatz].COUNT(< Erläuterung : STRING,
Vorzeichen: STRING,
Ansatz : ANSATZ >, Faktor : FLOAT),

Abbildung 5.14 stellt die erzeugten Objekte dar, die in gleicher Weise in ein Dokument des AVA-Systems eingefügt werden können. Die Objekte unterliegen dabei folgenden Typinformationen:

RenameAVA_Position = $\mathcal{K}_T$({<Erläuterung,String,normal> ,
<MAnsatz, $\mathcal{K}_S$($\mathcal{K}_T$ ({ <Erläuterung, String, normal>,
<Vorzeichen, String, normal>,
<Faktor, Float, normal>,
<Ansatz, String, normal>
})),normal>
}).

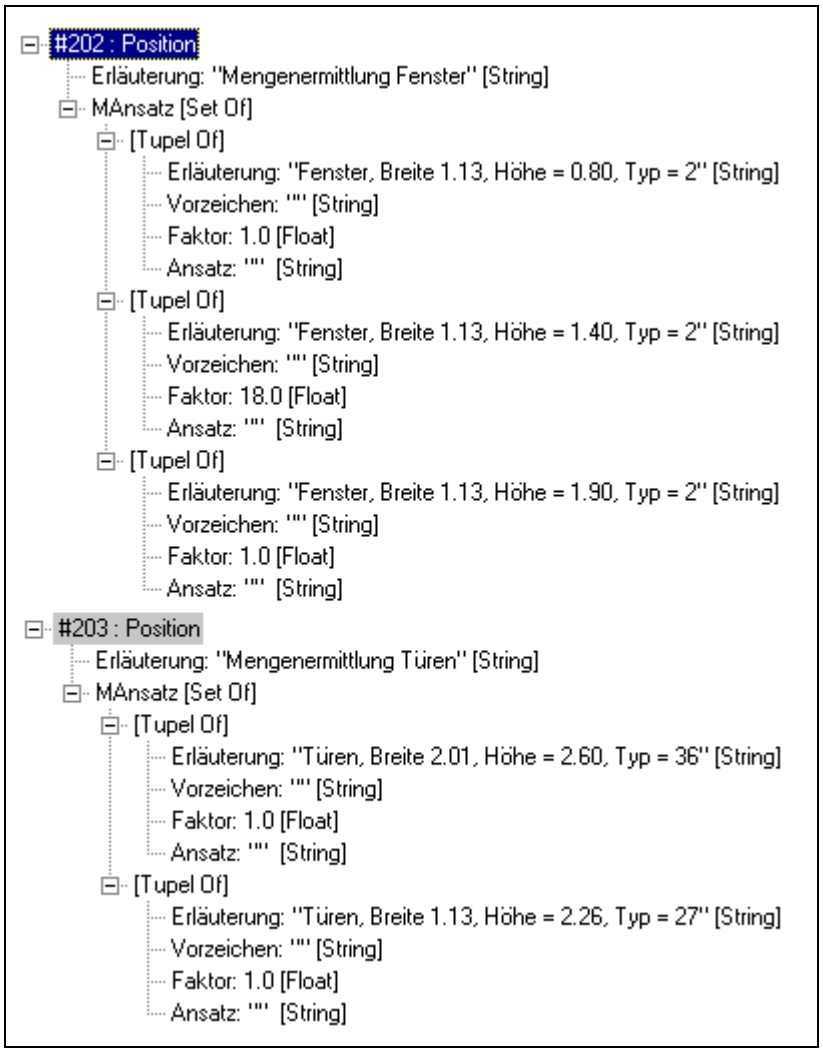


Abbildung 5.14: Objekte des Auswahlsatzes *AVA_Position*

5.2.3 Mengenermittlung der Stürze

Dieser Abschnitt dokumentiert die Möglichkeit des Integrationsmoduls, Mengendaten zu ermitteln, die nicht explizit in der zugrundeliegenden Planungsunterlage modelliert wurden. Vertreter hierfür sind Stürze, die Öffnungen der Wände abdecken müssen und die sich somit innerhalb der Wände befinden.

Für die Mengenermittlung dieser Stürze ist die Länge der Öffnung ausschlaggebend, die auch als klassifizierendes Merkmal für den Mengenansatz verwendet wird. Des weiteren bestimmt die Dicke einer Wand die Anzahl der einzubauenden Stürze.

Demnach muß für alle Öffnungen, die sich beispielsweise durch den Einbau von Fenstern und Türen ergeben, die Länge ermittelt werden. Diese Längen müssen um die definierte Auflagerlänge erweitert werden, die üblicherweise im Mauerwerksbau 10 cm [Schnei92] betragen. Je nach Wanddicke müssen entweder bei einer 24er Wand zwei Stürze pro Öffnung oder drei Stürze bei

einer 36er Wand angegeben werden. Für die Darstellung der Mengen im AVA-System ist die Anzahl der benötigten Stürze ausschlaggebend.

Die Übernahme beruht demnach auf folgenden Schritten:

- Erzeugen eines Auswahlsatzes bestehend aus allen Wandobjekten. Jedes Wandobjekt assoziiert die zur Wand gehörenden Fenster- und Türöffnungen. Die Attribute der Wand ergeben sich aus *ID*, *Thickness*, *Windows* und *Doors*, wobei die letzten beiden Attribute um die für den Mengenansatz benötigten Attribute *Erläuterung* und *Ansatz* erweitert werden.
  Name: WandÖffnungen.

- Auswahl aller 36er Wandobjekte.
  Name: SelectWände_36

- Setzen der Attribute *Ansatz* auf den Wert 3, Erläuterung auf den Wert 'Sturz in einer 36er Wand'.
  Name: UpdateWände_36

- Auswahl aller 24er Wandobjekte.
  Name: SelectWände_24

- Setzen des Attributes *Ansatz* auf den Wert 2, Erläuterung auf den Wert 'Sturz in einer 24er Wand'.
  Name: UpdateWände_24

- Erzeugen eines neuen Auswahlsatzes *Stürze_36* basierend auf dem Auswahlsatz *UpdateWände_36* durch die Umwandlung der beiden Mengenattribute *Windows* und *Doors* in einen Referenztyp und durch das Vereinigen beider resultierender Auswahlsätze.
  Name: CreateStürze_36

- Erzeugen eines neuen Auswahlsatzes *Stürze_24* basierend auf dem Auswahlsatz *UpdateWände_24* durch die Umwandlung der beiden Mengenattribute *Windows* und *Doors* in einen Referenztyp und durch das Vereinigen beider resultierender Auswahlsätze.
  Name: CreateStürze_24

- Vereinigen beider Auswahlsätze *Stürze_36* und *Stürze_24* zu einem Auswahlsatz.
  Name: AlleStürze

- Generieren eines neuen Auswahlsatzes, deren Objekttyp sich aus den beiden Attributen *Erläuterung* und *MAnsatz* zusammensetzt, wobei *MAnsatz* die Attribute *Erläuterung* vom Typ *String* und *MAnsatz* vom Typ *Reference To* ist.
  Name: CreateAVA_Position

- Einfügen eines neuen Objektes in den Auswahlsatz *CreateAVA_Position*, wobei das Attribut *Erläuterung* mit der Zeichenkette 'Stürze für Fenster und Türen' initialisiert wird und das Attribut *MAnsatz* mit den Objekten des Auswahlsatzes *AlleStürze* verbunden wird.
  Name: InsertPosition

- Umwandeln des Datentyps des Attributes *MAnsatz* in einen Mengendatentyp.
  Name: UpdateMAnsatz

- Aufaddieren aller Stürze mit der gleichen Länge.
  Name: AggregationCount
- Manipulation der einzelnen Längen auf die gewünschten durch das Einarbeiten der Auflagerlängen und das Runden auf handelsübliche Werte.
  Name: AVA_Position

Mit den folgenden Kommandos kann die Mengenermittlung der Stürze realisiert werden.

Kommando: **Name:** AlleInnenwände

Innenwand_24.UNION (*Innenwand_36* , < ID : LONG >) ;

Kommando: **Name:** AlleWände

AlleInnenwände.UNION (*Außenwand_36* < ID : LONG >);

Die Manipulation der Objekttypen *Window* und *Door* bedingt die Umwandlung der Attribute *Windows* und *Doors* in einen Mengentyp.

Kommando: **Name:** UpdateRefs

AlleWände.UPDATE ([Windows], [Doors]
SET
  [Windows].CREATESET(),
  [Doors].CREATESET()
) ;

Kommando: **Name:** ProjektionAttribute

UpdateRefs.PROJECTION (ID : LONG,
Thickness : FLOAT,
Windows : SET OF [TUPEL OF < ID : LONG, Width : FLOAT >],
Doors : SET OF [TUPEL OF < ID : LONG, Width : FLOAT >]
) ;

Kommando: **Name:** WandÖffnungen

ProjektionAttribute.EXTEND (Windows : SET OF [TUPEL OF < Erläuterung : STRING,
Ansatz: STRING >] ,
Doors : SET OF [TUPEL OF < Erläuterung : STRING,
Ansatz: STRING >]

) ;

Die folgenden Kommandos erstellen einen neuen Auswahlsatz, der alle Stürze mit den für den Mengenansatz geforderten Attribute beinhaltet.

Kommando: **Name:** SelectWände_36

WandÖffnungen.SELECT ([Thickness] = 0.36) ;

Kommando: **Name:** UpdateWände_36

SelectWände_36.UPDATE ([Windows],[Doors] SET
[Windows].UPDATE(USES [Width]
SET < [Ansatz] = '3',
[Erläuterung] = 'Stürze in einer 36er Wand, Länge = ' + ([Width]+0.20) >
) ,
[Doors].UPDATE(USES [Width]
SET < [Ansatz] = '3',
[Erläuterung] = 'Stürze in einer 36er Wand, Länge = ' + ([Width]+0.20)
)
) ;

Kommando: **Name:** UpdateSet_36

UpdateWände_36.UPDATE ([Windows],[Doors]
SET
[Windows].CREATEREFERENCE(FensterStürze_36) ,
[Doors].CREATEREFERENCE(TürStürze_36) ,
) ;

Kommando: **Name:** SelectWände_24

WandÖffnungen.SELECT ([Thickness] = 0.24) ;

Kommando: **Name:** UpdateWände_24

SelectWände_24.UPDATE ([Windows],[Doors] SET
[Windows].UPDATE(USES [Width]
SET < [Ansatz] = '2',
[Erläuterung] = 'Stürze in einer 24er Wand, Länge = ' + ([Width]+0.20) >
) ,
[Doors].UPDATE(USES [Width]
SET < [Ansatz] = '2',
[Erläuterung] = 'Stürze in einer 24er Wand, Länge = ' + ([Width]+0.20) >
)
) ;

Kommando: **Name:** UpdateSet_24

UpdateWände_24.UPDATE ([Windows],[Doors]
SET
[Windows].CREATEREFERENCE(FensterStürze_24) ,
[Doors].CREATEREFERENCE(TürStürze_24) ,
) ;

Das folgende Kommando vereinigt alle vier Auswahlsätze, die durch die Umwandlung der Mengenattribute *Windows* und *Doors* in Referenztypen jeweils für die 24er und 36er Wände gewonnen wurden. Das Kommando besitzt folgenden Aufbau:

Kommando: **Name:** AlleStürze

```
FensterStürze_36.UNION (
   TürStürze_36.UNION (
      FensterStürze_24.UNION ( TürStürze_24, < ID : LONG > )
   , < ID : LONG > )
,< ID : LONG > ) ;
```

Kommando: **Name:** CreateAVA_Position

```
CREATE ( Erläuterung: STRING, MAnsatz: REFERENCE TO() ) ;
```

Kommando: **Name:** InsertPosition

```
CreateAVA_Position.INSERT (
   UPDATE [Erläuterung],[MAnsatz]
   SET
      [Erläuterung] = 'Stürze für Fenster und Türen',


      [MAnsatz].Set ( AlleStürze )
) ;
```

Kommando: **Name:** UpdateMAnsatz

```
InsertPosition.UPDATE ( [MAnsatz] SET [MAnsatz].CREATESET() );
```

Kommando: **Name:** AVA_Position

```
UpdateMAnsatz.UPDATE ( [MAnsatz] SET
   [MAnsatz].COUNT ( TUPEL OF < Erläuterung : STRING,
                                Vorzeichen: STRING,
                                Ansatz : STRING >, Faktor : FLOAT )
) ;
```

Die resultierende Datenstruktur besitzt den gleichen Aufbau wie die erzeugte Datenstruktur der Mengenübernahme der Fenster und der Türen auf Seite 139. Abbildung 5.15 stellt die erzeugte Position mit den übernehmbaren Daten dar.

5.3 Bemessung von Unterzügen

Das folgende Beispiel soll noch einmal die Flexibilität des Datenaustausches unterstreichen, die durch die Anwendung des Integrationsmoduls möglich wird und ausschließlich durch den Anwender gesteuert wird. Diskutiert werden sollen verschiedene Abbildungsvorschriften für Unterzüge, die ausgehend von gleichen CAD-Objekten verschiedene Statik-Objekte erzeugen.

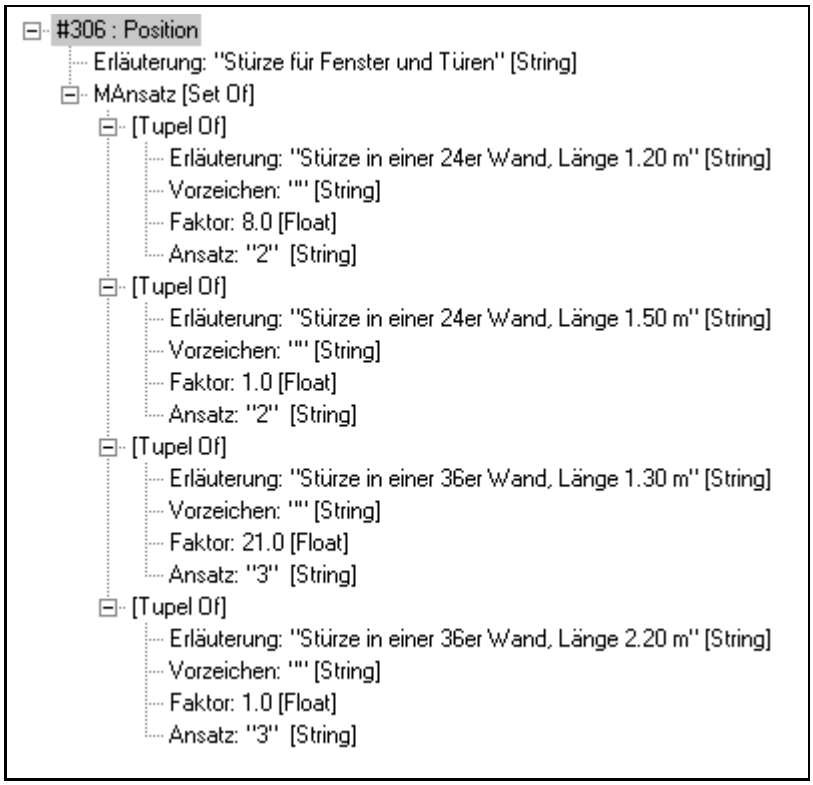


Abbildung 5.15: Position für die Mengenermittlung der Stürze im Integrationsmodul

Die Abbildungsvorschriften der Kategorie I erzeugen aus den CAD-Objekten des Datentyps UnterÜberzug gleichartige Statikobjekte. Die Statikobjekte sind unabhängig von der Decke und unterhalb dieser angeordnet. Im Gegensatz zu diesen Statikobjekten behandeln die Abbildungsvorschriften der Kategorie II die Erzeugung von Unterzugobjekten für die Statik, die die Deckendicke als Druckzone mitverwenden. Dabei verändert sich jedoch nicht die Breite des Unterzuges. Die Erzeugung der Unterzüge mit mittragender Deckenplatte wird in der letzten Kategorie besprochen. Abbildung 5.16 zeigt noch einmal die grafischen Präsentationen der Ausgangsobjekte und der resultierenden Statik-Objekte, die durch die Anwendung der jeweiligen Abbildungsvorschriften einer Kategorie erzeugt werden sollen sowie die semantische Bedeutung der einzelnen Attribute.

- Abbildungsvorschriften der Kategorie I

Diese Abbildungsvorschriften haben zum Ziel, die übernommene Geometrie auf die Attribute *Height*, *Thickness* und *Length* abzubilden und eine Ersetzung der Attributnamen in h, d und *Länge* durchzuführen.

Das folgende Kommando wählt alle Objekte aus dem Auswahlsatz *UnterÜberzüge* aus, die vom Typ der Modellierung Unterzüge sind. Hierzu muß das Attribut *Überzug* betrachtet werden, welches durch die Auswertung des Attributwertes die Entscheidung ermöglicht, ob es sich bei dem betrachteten Objekt um einen Unterzug handelt oder nicht.

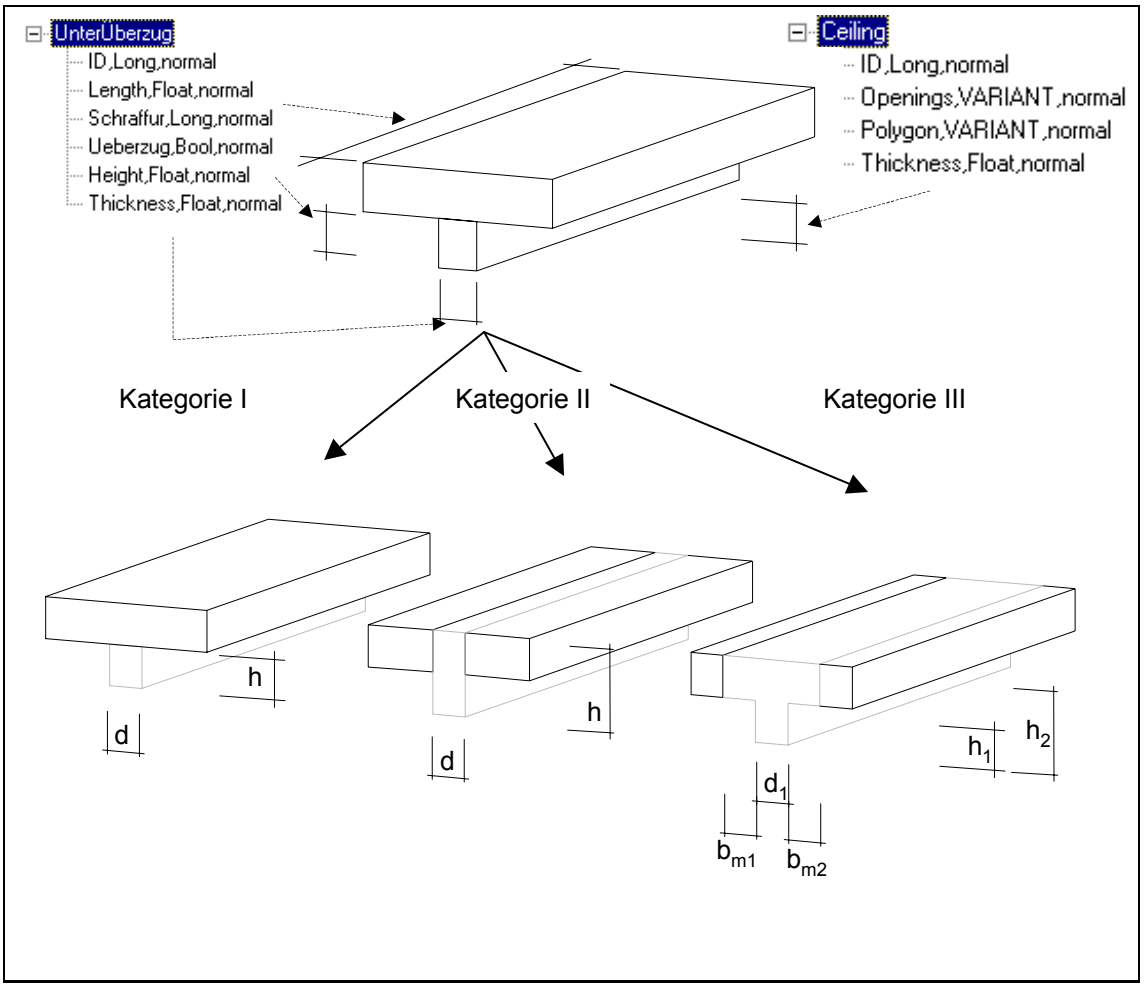


Abbildung 5.16: Beschreibung der verwendeten Datentypen

Kommando: **Name:** Unterzüge

Das folgende Kommando projiziert den Datentyp eines Unterzuges auf die gewünschten Attribute:

UnterÜberzüge.SELECT ([Überzug] = FALSE) ;

Kommando: **Name:** ProjektionQuader

Unterzüge.PROJECTION (Height : FLOAT,
Thickness : FLOAT,
Length : FLOAT
) ;

Wie für die Stürze müssen bei den Unterzügen noch die Auflagerlängen zu den modellierten Längen hinzugefügt werden. Für dieses Beispiel sollen Auflagerlängen von 15 cm auf beiden Seiten betrachtet werden.

Kommando: **Name: UpdateLength**

ProjektionQuader.UPDATE ([Length] SET [Length] + = 0.30) ;

Die folgenden drei Kommandos führen die Umbennenung der Attributnamen durch:

Kommando: **Name: RenameHeight**
UpdateLength.RENAME([Height] ALIAS h) ;

Kommando: **Name: RenameThickness**
RenameHeight.RENAME([Thickness] ALIAS d) ;

Kommando: **Name: RenameLength**
RenameThickness.RENAME([Height] ALIAS Länge) ;

- Abbildungsvorschriften der Kategorie II

Die Abbildungsvorschriften dieser Kategorie müssen noch die Höhe des Unterzuges um die Höhe der Decke erweitern. Hierzu sind drei zusätzliche Kommandos notwendig. Das erste Kommando kombiniert die Objekte des Auswahlsatzes *Unterzüge* mit dem einzigen Objekt des Auswahlsatzes *Decke*. Damit das Deckenattribut *Thickness* bei der Kombination erhalten bleibt, muß das Attribut umbenannt werden. Das letzte Kommando manipuliert die Höhe des Unterzuges. Die neue Höhe des Unterzuges ergibt sich aus der Addition der alten Höhe des Unterzuges mit der Höhe der Decke.

Kommando: **Name:** RenameThickness

Ceiling.RENAME ([Thickness] ALIAS DickeDecke) ;

Kommando: **Name:** JoinUnterzugMitDecke

Unterzüge.JOIN (*RenameThickness*) ;

Kommando: **Name:** UpdateHeight

JoinUnterzugMitDecke.UPDATE ([Height] USES [DickeDecke] SET [Height] += [DickeDecke]) ;

Die folgenden Kommandos sind den unter Kategorie I angegebenen Kommandos ähnlich. Diese Kommandos haben zum Ziel, die möglichen Attribute des entstandenen Objekttyps auf die notwendigen Attribute einzuschränken, die Länge um die Auflagerlänge zu erweitern und anschließend eine Namensersetzung durchzuführen. Die fehlenden Kommandos lauten:

Kommando: **Name: ProjektionQuader**

UpdateHeight.PROJECTION (Height : FLOAT,
Thickness : FLOAT,
Length : FLOAT
) ;

Kommando: **Name: UpdateLength**

ProjektionQuader.UPDATE ([Length] SET [Length] + = 0.30) ;

Kommando: **Name: RenameHeight**
UpdateLength.RENAME([Height] ALIAS h) ;

Kommando: **Name: RenameThickness**
RenameHeight.RENAME([Thickness] ALIAS d) ;

Kommando: **Name: RenameLength**
RenameThickness.RENAME([Height] ALIAS Länge) ;

- Abbildungsvorschriften der Kategorie III

Für die Abbildung der Attribute *d1*, *h1* und *h2* werden analoge Abbildungsvorschriften benötigt. Die einzelnen Unterzüge müssen mit dem entsprechenden Deckenobjekt kombiniert werden. Die Berechnung der Attributwerte erfordert auch keine weitere Erläuterung, genauso die Projektion auf die gewünschten Attribute. Diese Kommandos sollen an dieser Stelle nicht noch einmal wiederholt werden.

Kompliziert ist die Berechnung der mitwirkenden Deckenbreite. Hierzu sind Kenntnisse erforderlich, die aus dem Gebiet der Statik kommen und von dem Anwender vorausgesetzt werden können. Demnach ist die mitwirkende Breite von zwei Verhältnissen abhängig. Das eine Verhältnis ergibt sich aus der Division von $(h_2 - h_1)/h_2$. Das zweite Verhältnis ergibt sich, wenn man die Hälfte der lichten Stützweite durch die Balkenlänge teilt. Bei Durchlaufträgern werden für die Balkenlänge die einzelnen Stützweiten angesetzt. Mit Hilfe dieser beiden Verhältnisse kann aus einer Tabelle ein Tafelwert interpoliert werden. Wird dieser Tafelwert mit der Hälfte der lichten Stützweite der Platte multipliziert, so erhält man die mitwirkende Breite der Platte auf der jeweiligen Seite des Unterzuges. Diese Methode für die Bestimmung der mitwirkenden Deckenbreite kann beispielsweise in [Schnei92] auf Seite 5.74 nachgelesen werden.

Der einfachste Weg für die Bestimmung der notwendigen Zahlenwerte ist der, daß man die Werte manuell[2] ermittelt, diese Werte den Attributen der Objekte zuweist, und nachdem die neuen Objekte im Statikdokument erzeugt worden sind, die fehlenden Beziehungen im Integrationsmodul angibt. Der Einsatz dieser manuellen Methode soll an dieser Stelle erwähnt werden, da diese Methode die Flexibilität des Integrationsmoduls unterstützt und somit ihre Berechtigung für eine einfache und praxisorientierte Datenübernahme besitzt.

Aus Gründen der Vollständigkeit soll für einen Unterzug (mit der Identität #55) die Berechnung der mittragenden Deckenbreite im Integrationsmodul dargestellt werden. Die Berechnung

[2] Für die Vermeidung von Eingabefehlern kann die Zwischenablage verwendet werden.

beruht auf der Erweiterung des dynamischen Funktionskonzeptes. Somit unterstützt ARCON eine ausgezeichnete Methode, die ausgehend von einem übergebenen ARCON-Objekt dessen grafische Ausdehnung berechnet. Das Resultat ist ein Rechteck, welches das Objekt umschließt. Diese primitive grafische Darstellung eines Objektes kann mit den vorhanden grafischen Operationen des Integrationsmoduls, die mit der Schnittstelle CIMO_GRAFIC eingebunden werden kann, weiter bearbeitet werden. Der Lösungsweg setzt sich aus folgenden Etappen zusammen:

- Übernahme der Objekte #55, #17 und #56 in jeweils einen Auswahlsatz mit den Namen *Unterzug*, *linkesTragelement* und *rechtesTragelement*,
- Abbildung des CAD-Unterzuges in einen Statik-Unterzug entsprechend den Anweisungen der Kategorie II,
- Erweiterung aller Objekttypen der Auswahlsätze *Unterzug*, *linkesTragelement* und *rechtesTragelement* um ein Attribut *BoundingBox* mit folgendem Typaufbau:
  $\mathsf{BoundingBox} = \mathcal{K}_T(\{<$ xLeft, Float, normal $>$
  $<$ yTop, Float, normal $>$
  $<$ xRight, Float, normal $>$
  $<$ yButtom, Float, normal $>$
  $\})$,
- Ermittlung der Ausdehnungen der einzelnen Objekte und Zuweisung der berechneten Werte zu dem zum Objekt gehörenden Attribut *BoundingBox*,
- Erweiterung des zum Auswahlsatz *Unterzug* gehörenden Datentyps um zwei Referenztypen *linkeWand* und *rechteWand*, sowie um die Attribute b_1, b_2, b_{m_1} und b_{m_2}, die alle den Datentyp *Float* unterstützen,
- Zuweisung der beiden neuen Attribute *linkeWand* und *rechteWand* zu den entsprechenden Auswahlsätzen *linkesTragelement* und *rechtesTragelement*,
- Umwandlung der beiden Attribute *linkeWand* und *rechteWand* in Mengentypen,
- Berechnung von b_1 und b_2,
- Berechnung der Werte b_{m_1} und b_{m_2} mittels einer weiteren, über das dynamische Funktionskonzept eingebundenen Funktion, die als Parameter die beiden Verhältnisse beinhaltet und die Länge der mittragenden Deckenplatte für die jeweilige Seite des Unterzuges zurückgibt.

5.4 Erläuterung der Aufgabenstellung: Wahrung der Konsistenz

Das letzte Beispiel dokumentiert die Arbeitsweise des Integrationsmoduls bei der Wahrung der Konsistenz zwischen den verschiedenen Planungsunterlagen. Die Beschreibung der Ausgangssituation nimmt hierbei eine dominante Rolle ein, da die implementierten Algorithmen und Datenstrukturen maßgeblich von den gewählten Aktionen abhängen.

Gegenstand des Beispieles ist es, den Anwender des AVA-Dokumentes über die aktuelle Veränderung des Architekturmodells zu informieren. Die Veränderungen ergeben sich nacheinander und befinden sich somit in zwei verschiedenen Freigabeständen des CAD-Modells. Der erste Freigabestand beinhaltet die Veränderung der Breite eines Fensters. Dieses Fenster wird im folgenden

Beschreibung	Definitionsbereich der Relation	Wertebereich der Relation
Mengenermittlung der Wand #22	#22, #23, #24, #25, #26, #27, #28	#106
Mengenermittlung der Fenster	#2, #3, #4, #5, #7, #8, #9, #10, #11, #12, #16, #18, #19, #20, #21, #23, #24, #25, #26, #27	#202
Mengenermittlung der Stürze	alle Fenster und Türen	#306

Tabelle 5.4: Modellierte Beziehungen basierend auf dem Fenster #24

Freigabestand gelöscht. Dabei wird davon ausgegangen, daß zwei verschiedene AVA-Dokumente aktualisiert werden müssen. Das eine Dokument beinhaltet die Wandmengen. Das andere Dokument die Menge aller Stürze, Fenster und Türen.

Erschwerend für die Verifikation des Beispiels soll folgende Situation angenommen werden. Nachdem der erste Freigabestand des CAD-Modells in das Integrationsmodul übernommen wurde, erfolgt die Aktualisierung der Wandmengen. Die restlichen Mengen bleiben zu diesem Zeitpunkt unberücksichtigt. Erst nachdem der zweite Freigabestand des CAD-Modells in das Integrationsmodul eingefügt wurde, werden beide AVA-Dokumente aktualisiert. Diese Situation entspricht dem ersten Sonderfall auf Seite 100.

Des weiteren beschreibt das Beispiel das Vorgehen bei der Erzeugung der zur Wahrung der Konsistenz gehörenden Objekte des Integrationsmoduls. Schrittweise wird der Ablauf dargestellt, der sich aus der Erstellung des Baumes, dessen Wurzel das veränderte Objekt und dessen restliche Knoten und Blätter die einzelnen abhängigen Objekte in den anderen Planungsunterlagen darstellen, die Verschmelzung des Baumes mit dem globalen Graphen des Integrationsmoduls, der die kompletten Informationen über den Zustand der Konsistenz beinhaltet und die Manipulationen der Relation CIMO_DEPENDENCE zusammensetzt.

Das Beispiel basiert auf den in den Abschnitten 5.2.1 bis 5.2.3 generierten Beziehungen. Festgelegt werden soll, daß es sich bei dem veränderten Fensterobjekt um das Fenster mit der Identifikation #24 handelt. Tabelle 5.4 zeigt noch einmal die modellierten Beziehungen zwischen CAD-Modell und den AVA-Modellen, in denen das ausgewählte Fenster auftritt.

Das Fallbeispiel kann somit in verschiedene Szenarien, die wiederholt ablaufen, aufgeteilt werden.

Des erste Szenario beschreibt die Aktualisierung eines gesamten Modells. Es erläutert die Erkennung von veränderten Objekten, das Aufstellen des globalen Graphen $\mathcal{G}$, wenn dieser zu diesem Zeitpunkt noch keine Elemente beinhaltet, sowie die Zuordnung der Knoten des Graphen zu den im Integrationsmodul enthaltenen Dokumenten.

Das zweite Szenario beinhaltet die Aktualisierung eines Objektes beispielsweise eines Fensters des Dokumentes 'Wandmengen'. Gegenstand des zweiten Szenarios ist es, die Möglichkeiten des Integrationsmoduls für die Ermittlung der zu aktualisierenden Objekte sowie der aufgetretenen Veränderungen effizient zu ermitteln, so daß der Anwender gezielt die Aktualisierung der Objekte durchführen kann. Des weiteren beinhaltet das Szenario die Beschreibung der einzelnen Prozesse im Integrationsmodul, die durch die Aktualisierung eines Objektes mittels des Integrationsmoduls angestoßen wird.

Das nächste Szenario ergibt sich aus der wiederholten Aktualisierung des CAD-Modells. Schwerpunkt hierbei ist es, die Verschmelzung des globalen Graphen $\mathcal{G}$ zu erläutern, der zu diesem Zeitpunkt bereits Knoten und Kanten beinhaltet. Zusätzlich wird die Bereinigung des Graphen dargestellt.

5.4.1 Erstes Szenario: Aktualisierung eines Modells

Die Ausgangssituation des Prozesses kann wie folgt beschrieben werden:

- Für alle Dokumente - das sind die beiden AVA-Dokumente 'Mengenermittlung Wände' und 'Mengenermittlung Stürze, Fenster und Türen' sowie das CAD-Dokument 'Grundriß' - existiert eine eingebettete Referenz im Integrationsmodul.
- Für jedes in den Abschnitten 5.2.1, 5.2.3 und 5.2.2 betrachtete Objekt wurde der Typ sowie der Zustand im persistenten Datenraum des Integrationsmoduls nachgebaut. Jedes Objekt erhielt dabei eine neue Identifikation.
- Die Instanz, die auf der Klasse CIMO_DEPENDENCE des Integrationsmoduls basiert, beinhaltet die Relationen der Tabelle 5.4.

Dieser Prozeß hat zum Ziel, die Ermittlung der Zustände für die einzelnen AVA-Dokumente durchzuführen. Zum Zeitpunkt vor der Aktualisierung des CAD-Modells beinhaltet der globale Graph $\mathcal{G}$ keine Elemente. Somit ergibt sich für alle Dokumente der Zustand *stark konsistent*.

Das Aktualisieren eines gesamten Dokumentes wird durch ein zum Standardumfang von OLE gehörendes Ereignis aktiviert. Das Ereignis steht in der Server-Applikation zur Verfügung und ermöglicht die persistente Speicherung des Server-Dokumentes im Client. parNach der Aktualisierung des CAD-Modells, die eine Manipulation von $\mathcal{G}$ mit sich bringt, muß sich der Zustand für alle AVA-Dokumente in *schwach konsistent* verändern. Die dafür notwendigen Schritte werden nun in den folgenden Abschnitten behandelt.

Entsprechend der Ausführung des Abschnittes 3.2.3 basiert der Prozeß auf drei Arbeitsschritten. Der erste Arbeitsschritt behandelt die Erkennung von veränderten Objekten, sobald ein neuer Freigabestand in das Integrationsmodul übernommen wird.

Der zweite Arbeitsschritt, der sich an den ersten Arbeitsschritt automatisch anschließt, führt die notwendigen Manipulationen an der im Integrationsmodul einmal angelegten Grapheninstanz durch, nachdem für jedes veränderte Objekt ein Abhängigkeitsbaum erzeugt wurde. Für die Erzeugung des Baumes werden die Resultate des ersten Arbeitsschrittes und die modellierten Relationen in der Instanz von CIMO_DEPENDENCE benötigt.

Für das gewählte Beispiel, in dem das Fenster #24 im CAD-Modell verändert wurde, würden die Mengen O^D und O^I keine Elemente beinhalten. Die Menge O^M würde ein Element besitzen und zwar das veränderte Fenster.

Für dieses Fenster wird nun ein Abhängigkeitsbaum generiert, dessen Wurzel das besagte Objekt ist. Abbildung 5.17 beschreibt die prinzipielle Datenstruktur des Baumes. Ihr prinzipieller Aufbau setzt sich aus folgendem Datentyp zusammen:

$$
\begin{aligned}
\mathit{Knoten} = \mathcal{K}_T(\{ &< \text{Ident, CIMO\_IDENT, normal}>, \\
&< \text{status, } \zeta\text{, normal}>, \\
&< \text{childs,}\mathcal{K}_L(\{\mathcal{K}_T(\{ < \text{Veränderung, } \kappa\text{, normal}>, \\
&\qquad\qquad < \text{Beachtet, } \lambda\text{, normal} >, \\
&\qquad\qquad < \text{Endpunkt, Knoten, normal} > \\
&\qquad\quad \}) \\
&\quad \}), \text{normal} > \\
\}).&
\end{aligned}
$$

Die Abarbeitung der Methode

CreateKnoten(#24, *aktuell*, *unbeachtet*, *modifiziert*)

erzeugt ein neues Wurzelobjekt und findet anschließend drei Relationen in der Instanz der Klasse CIMO_DEPENDENCE. Der resultierende Teilbaum besitzt folgende Attributwerte:

wurzel = < Ident : #24 [CIMO_IDENT] ,
status : *aktuell* [ζ] ,
child : { < Veränderung: *modifiziert* [κ] ,
Beachtet : *unbeachtet* [λ] ,
Endknoten: < Ident : #106 [CIMO_IDENT],
status : *aktualisierbar* [κ],
child : EMPTY [$\mathcal{K}_L$]
> [Knoten]
> ,
< Veränderung: *modifiziert* [κ] ,
Beachtet : *unbeachtet* [λ] ,
Endknoten: < Ident : #202 [CIMO_IDENT],
status : *aktualisierbar* [κ],
child : EMPTY [$\mathcal{K}_L$]
> [Knoten]
> ,
< Veränderung: *modifiziert* [κ] ,
Beachtet : *unbeachtet* [λ] ,
Endknoten: < Ident : #306 [CIMO_IDENT],
status : *aktualisierbar* [κ],
child : EMPTY [$\mathcal{K}_L$]
> [Knoten]
>
} [$\mathcal{K}_L$]
>

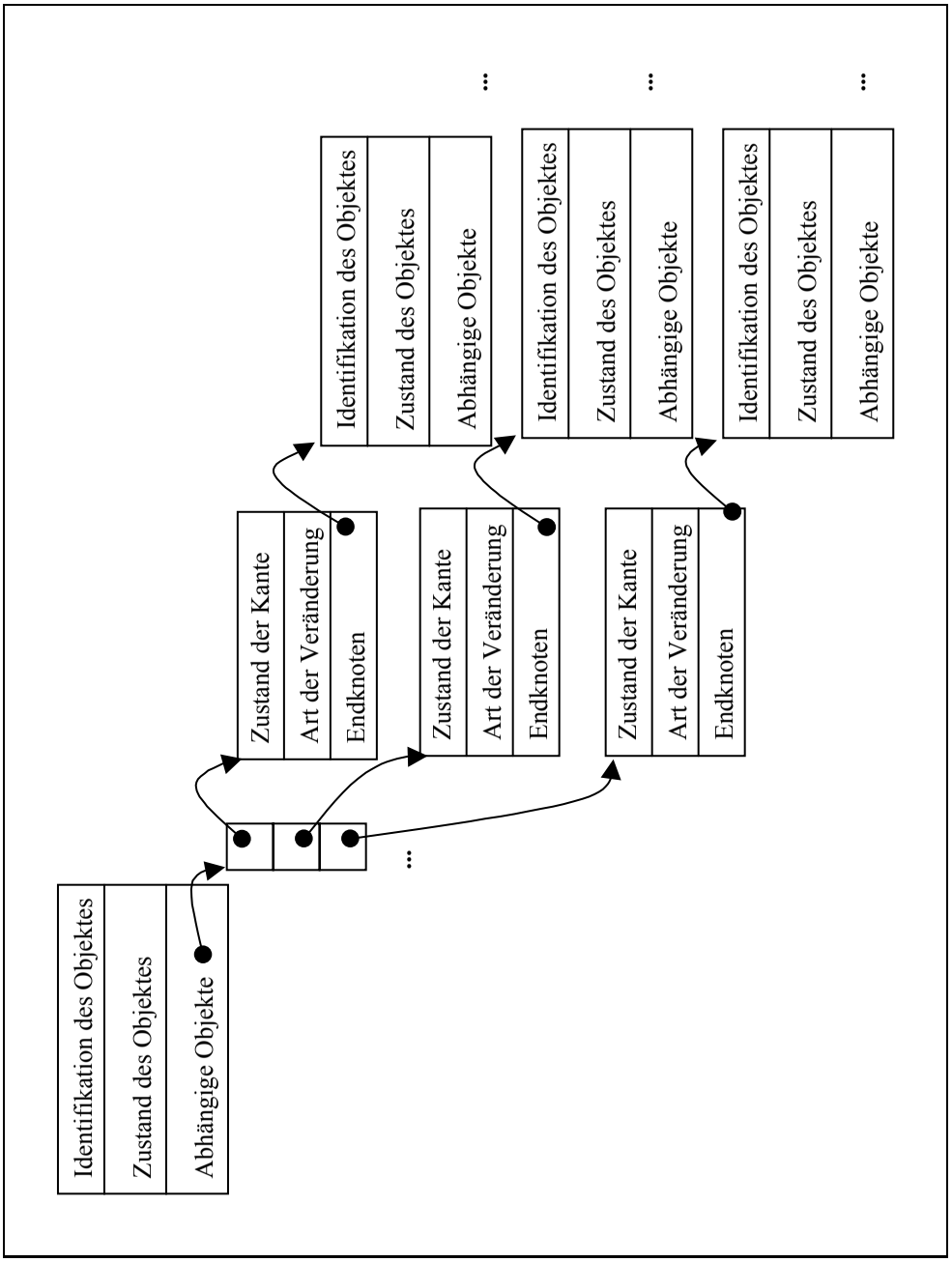


Abbildung 5.17: Strukturelle Zusammensetzung des Abhängigkeitsbaumes

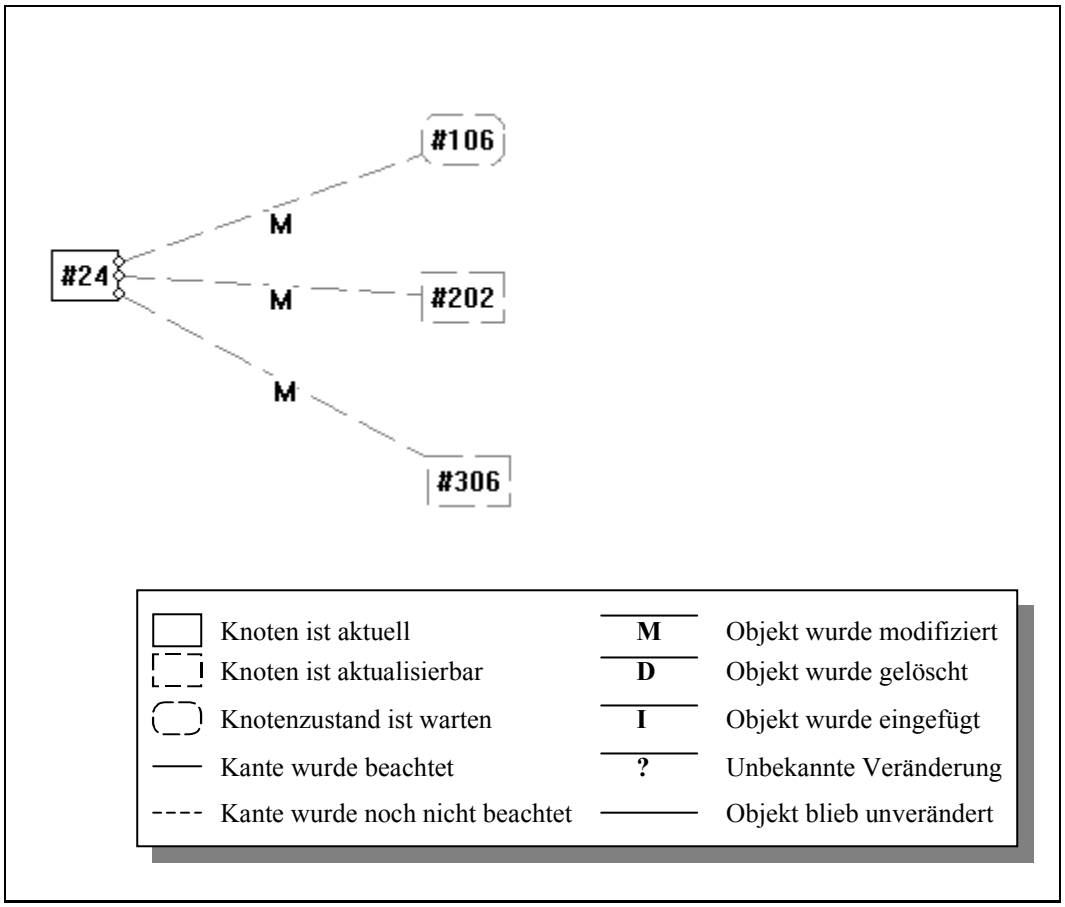


Abbildung 5.18: Erzeugter Abhängigkeitsgraph

Der erzeugte Baum wird nun im anschließenden Schritt in den Graphen $\mathcal{G}$ sequentiell eingearbeitet. Die Kanten des Baumes werden dabei sequentiell entsprechend dem Algorithmus einer Tiefensuche ermittelt. Die Attributwerte werden entsprechend den Tabellen 4.5 und 4.6 auf der Seite 107 manipuliert. Die Abbildung 5.18 zeigt den erzeugten Graphen, nachdem das veränderte Objekt in den Graphen eingearbeitet wurde.

Der erzeugte Graph, der auf dem gewählten Beispiel aufbaut, besitzt zwei Besonderheiten. Zum einen beinhaltet die zugehörige Instanz der Datenstruktur zu diesem Zeitpunkt noch keine Elemente, und zum anderen umfaßt das Beispiel nur die Veränderung eines Objektes.

Der letzte Arbeitsschritt umfaßt die Zerlegung aller im Graphen enthaltenen Knoten in Äquivalenzklassen, wobei das Kriterium für die Zerlegung die Zugehörigkeit eines Objektes zu einem eingebetteten Dokument ist. Die Zugehörigkeit eines Objektes zu einem Dokument ergibt sich aus der Auswertung der Objektidentifikation, die mittels einem Attribut zu jedem Knoten angegeben wurde.

Die Zerlegung führt automatisch eine Aktualisierung der Darstellung von den einzelnen Dokumenten in der Benutzeroberfläche des Integrationsmoduls durch. Ziel hierfür ist, den einzelnen Anwender über eine Veränderung des Konsistenzzustandes zu informieren.

Für das gewählte Beispiel ergibt die Zerlegung des Graphen die Zuordnung des Objektes mit der Identifikation #106 dem Dokument 'Wandmengen' und für die Objekte mit den Identifikationen #202 und #306 das Dokument 'Mengenermittlung der Fenster, Türen und Stürze'. Demnach werden diese beiden Dokumente im Integrationsmodul hervorgehoben.

Der nächste Abschnitt befaßt sich gezielt mit den Möglichkeiten des Anwenders, die inkonsistenten Objekte aufzufinden und entsprechend der vorliegenden Veränderung geeignete Aktionen zu wählen, um eine Anpassung des Dokumentes durchzuführen.

5.4.2 Zweites Szenario: Aktualisierung von Objekten

Der Eigentümer des AVA-Dokumentes 'Mengenermittlung' erkennt anhand der Markierung in der Benutzeroberfläche des Integrationsmoduls, daß sein Dokument nicht mehr konsistent zu den anderen im Integrationsmodul eingebetteten Dokumenten ist. Das Integrationsmodul erstellt nach der entsprechenden Aufforderung durch den Anwender eine Zusammenstellung der Objekte, die Elemente des gewählten Dokumentes sind und die überprüft und gegebenenfalls aktualisiert werden müssen.

Für jedes Objekt, welches einer Aktualisierung bedarf, kann der Anwender in einem zusätzlichen View detaillierte Informationen angezeigt bekommen. Abbildung 5.19 zeigt exemplarisch das Objekt #106 des gewählten Dokumentes sowie die abhängigen Objekte aus dem Dokument 'Grundriß'. Die Optionen für die Konfiguration des Views wurden im Abschnitt 3.2.3 auf Seite 53 erläutert.

Zusätzlich kann der Anwender Objekte des Views auswählen und durch eine entsprechende Aktion diese Objekte in den zugrunde liegenden Dokumenten hervorgehoben betrachten. Diese Möglichkeit gehört zu dem standardisierten Kommunikationsumfang des Integrationskonzeptes.

Nachdem der Eigentümer sich intensiv mit den aufgetretenen Veränderungen befaßt hat, erfolgt die Aktualisierung seines Dokumentes. Grundsätzlich stehen den Anwendern dabei zwei Möglichkeiten der Aktualisierung offen. Die erste Möglichkeit ergibt sich aus den gleichen Arbeitsschritten wie sie im Abschnitt 5.2.1 vorgestellt wurden und soll in diesem Szenario besprochen werden. Die zweite Möglichkeit erlaubt den Anwendern, die Aktualisierung des Dokumentes mit der Funktionalität durchzuführen, die die entsprechende, das Dokument verarbeitende Applikation dem Anwender zur Verfügung stellt. Dieses Szenario wurde im Abschnitt 5.4.1 detailliert behandelt.

Ausgangspunkt für die Aktualisierung bildet wiederum ein neuer Auswahlsatz, der durch die Selektion der Wand #22 erzeugt wird. Entsprechend den Ausführungen des Abschnittes 5.2.1 werden durch die Auswahl der Wand drei Auswahlsätze im Integrationsmodul generiert.

Die entsprechenden Kommandos (siehe Abschnitt 5.2.1) erzeugen eine neue Position für das AVA-Dokument 'Wandmengen', deren Attributwerte im anschließenden Schritt der existierenden Position #106 im AVA-Dokument 'Wandmengen' zugewiesen werden. Dafür stellt das Integrationsmodul ein separates Ereignis zur Verfügung.

Das entsprechende Ereignis für die Aktualisierung einzelner Objekte besitzt im Gegensatz zum Einfügen eines neuen Objektes die Möglichkeit, eine Verbindung zwischen den Quellobjekten und den Zielobjekten anzugeben und somit dem Zielobjekt die gewünschten Attributwerte des

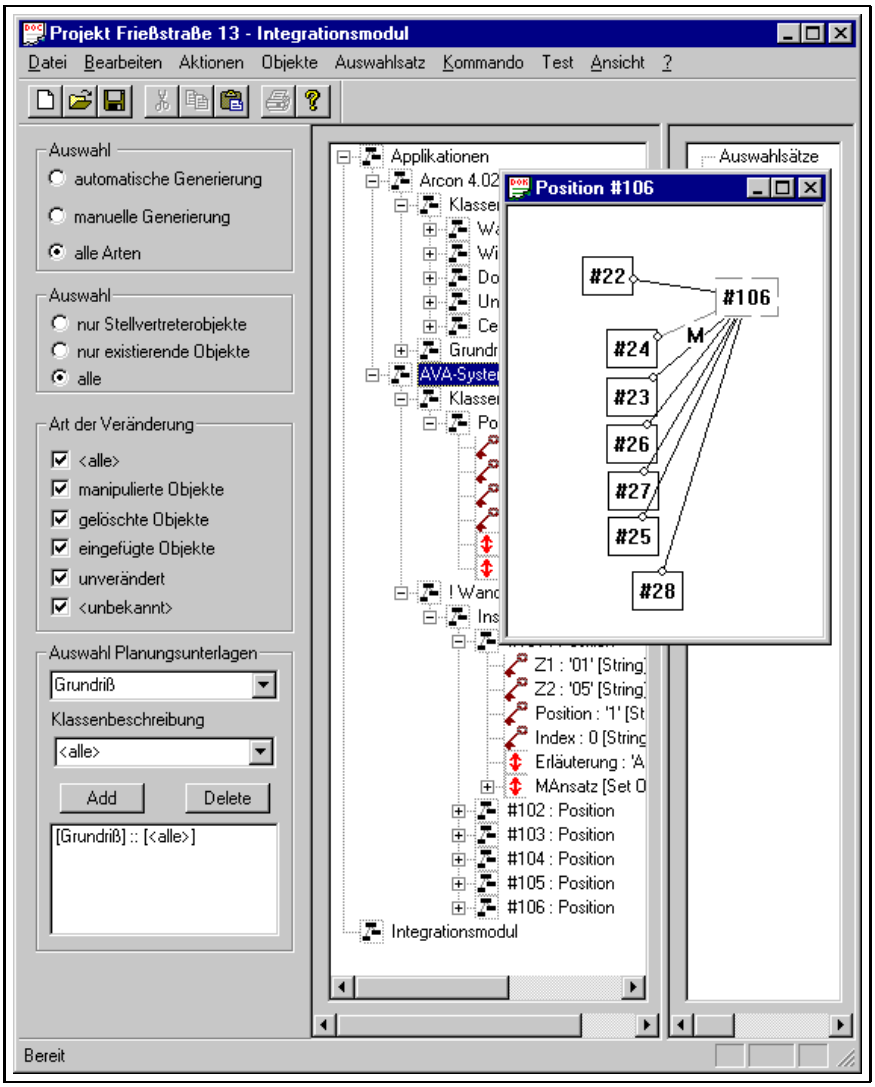


Abbildung 5.19: Darstellung von Beziehungen

Quellobjektes zuzuweisen. Hierzu wird erneut der Anwender aufgefordert, die zu aktualisierenden Objekte durch die Auswahl eines Auswahlsatzes, welcher die gewünschten Objekte beinhaltet, auszuwählen. Anschließend muß der Anwender die Abbildungsvorschriften für die Objekttypen angeben (vergleiche hierzu Abbildung 5.12 auf Seite 134) und die Übertragung starten.

Dem Integrationsmodul kommen zwei Aufgaben zu. Die erste Aufgabe umfaßt das Finden des richtigen Objektes im Zielsystem in Abhängigkeit eines aktuellen Quellobjektes im dynamischen Datenraum des Integrationsmoduls. Die zweite Aufgabe muß sicherstellen, daß das Integrationsmodul die Objekte im persistenten Datenraum des Integrationsmoduls mit den aktualisierten Werten ablegt.

Die Ausgangssituation der ersten Aufgabe ergibt sich, indem folgende Informationen bereitgestellt werden:

- ein angegebener Auswahlsatz, der die Quellobjekte beinhaltet, die für die Aktualisierung verwendet werden sollen,
- ein ausgewähltes Dokument, welches die Zielobjekte enthält und
- eine Menge von Abbildungsvorschriften, die die Zuweisung der veränderten Attributwerte angeben.

Für jedes Objekt im angegebenen Auswahlsatz lassen sich die Zielobjekte finden, indem die Relationen der Instanz CIMO_DEPENDENCE ausgewertet werden. Die Quellobjekte beinhalten die persistenten Objektidentifikatoren von den Objekten, die für die Erzeugung der Quellobjekte verwendet wurden. Das gesuchte Zielobjekt muß von den gleichen Objekten abhängen und muß demnach persistente Tupel in der Instanz der Klasse CIMO_DEPENDENCE besitzen, deren Definitionswerte den persistenten Objektidentifikationen des Quellobjektes entsprechen.

In dem Fall, daß das Integrationsmodul mehrere Objekte findet, die die angegebenen Bedingungen erfüllen oder daß das Integrationsmodul kein Objekt findet, das die angegebenen Bedingungen erfüllt, muß der Anwender die Zuordnung selbst vornehmen.

Anderenfalls wird der Anwender aufgefordert, die Aktualisierung der gefundenen Objekte zu bestätigen.

Nach dem Aktualisieren des Objektes im Dokument schließen sich automatisch die Aktualisierung des Objektes im persistenten Datenraum des Integrationsmoduls sowie die Aktualisierung der Objektbeziehungen an. Die Objektidentifikation von jedem aktualisierten Objekt wird in die Menge O^M eingetragen, so daß nach diesem Prozeß wieder alle abhängigen Objekte ermittelt werden können und der Zustand des Graphen $\mathcal{G}$ die Veränderungen ausdrückt.

Da das Objekt #106 kein Basisobjekt im Gesamtmodell des Integrationsmoduls darstellt, behält der Graph $\mathcal{G}$ zu diesem Zeitpunkt seinen Zustand bei.

Zusätzlich verändert dieser Prozeß noch das Kantenattribut λ auf *beachtet* von allen Kanten, deren Endknoten eines der veränderten Objekte des ersten Auswahlsatzes sind. Wie im Abschnitt 4.2.5 beschrieben, läßt sich dieser Schritt damit begründen, daß alle abhängigen Objekte für die Aktualisierung verwendet und somit auch beachtet wurden.

Für das gewählte Beispiel besitzt der Graph den in Abbildung 5.20 dargestellten Zustand.

5.4.3 Drittes Szenario: Bearbeitung des Graphen

Ausgangspunkt für die Betrachtung des dritten Szenarios bilden die Tätigkeiten:

1. Aktualisierung des Dokumentes 'Wandmengen', wobei das Fenster mit der Identifikation #24 in Bezug auf seine Öffnungsmaße verändert wurde.
2. Aktualisierung des Objektes mit der Identifikation #106 im Dokument 'Wandmengen' bezüglich des veränderten Mengenanteils, der sich aus den veränderten Öffnungsmaßen des Fensters ergibt.

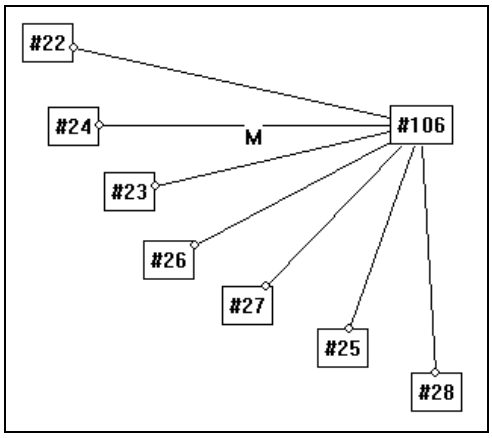


Abbildung 5.20: Resultierender Zustand des Graphen bezüglich des Objektes #106

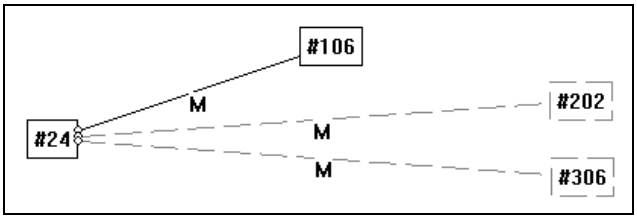


Abbildung 5.21: Zustand des Graphen nach dem zweiten Szenario

3. Aktualisierung des Dokumentes 'Wandmengen', wobei zu diesem Zeitpunkt das Fenster mit der Identifikation #24 gelöscht wurde.

4. Aktualisierung aller Dokumente des AVA-Systems.

Nach dem ersten Arbeitsschritt beinhaltet der Graph $\mathcal{G}$ den in Abbildung 5.18 dargestellten Zustand. Abbildung 5.21 zeigt den Zustand des Graphen nach dem Aktualisieren des Objektes mit der Identifikation #106.

Das Aktualisieren des Dokumentes 'Wandmengen' führt erneut zur Generierung eines Abhängigkeitsbaumes, dessen Knoten wieder das Objekt mit der Identifikation #24 ist. Da das Objekt im Dokument 'Wandmengen' nicht mehr enthalten ist, wurden das Objekt sowie die Relationen, die dieses Objekt verwenden, aus der Instanz CIMO_DEPENDENCE entfernt. Im folgenden wird der Zustand des Baumes wiedergegeben:

wurzel = < Ident : #24 [CIMO_IDENT] ,
status : *aktuell* [ζ] ,
child : { < Veränderung: *gelöscht* [κ] ,
Beachtet : *unbeachtet* [λ] ,
Endknoten: < Ident : #106 [CIMO_IDENT],
status : *aktualisierbar* [κ],
child : EMPTY [$\mathcal{K}_L$]
> [Knoten]
> ,
< Veränderung: *gelöscht* [κ] ,

Bezeichnung der Kante	alter Zustand	neuer Zustand
#24→#106	*modifiziert* *beachtet*	*gelöscht* *unbeachtet*
#24→#202	*modifiziert* *unbeachtet*	*gelöscht* *unbeachtet*
#24→#306	*modifiziert* *unbeachtet*	*gelöscht* *unbeachtet*
#22→#306 #22→#202 #22→#306		*modifiziert* *unbeachtet*

Tabelle 5.5: Überführung der Kantenattribute des Graphen

Beachtet : *unbeachtet* [λ] ,
Endknoten: < Ident : #202 [CIMO_IDENT],
status : *aktualisierbar* [κ],
child : EMPTY [$\mathcal{K}_L$]
> [Knoten]
> ,
< Veränderung: *gelöscht* [κ] ,
Beachtet : *unbeachtet* [λ] ,
Endknoten: < Ident : #306 [CIMO_IDENT],
status : *aktualisierbar* [κ],
child : EMPTY [$\mathcal{K}_L$]
> [Knoten]
>
} [$\mathcal{K}_L$]
>

Des weiteren wird nach der Aktualisierung des Dokumentes 'Wandmengen' noch ein weiteres Objekt gefunden, welches durch das Löschen des Fensters manipuliert wurde. Es handelt sich dabei um das Wandobjekt mit der Identifikation #22, da die Anzahl der referenzierten Fensterobjekte verändert wurden. Auch für dieses Objekt wird ein ähnlicher Abhängigkeitsbaum erzeugt.

Der Algorithmus für die Verschmelzung der Abhängigkeitsbäume mit dem Graphen $\mathcal{G}$ muß jetzt den bestehenden Zustand des Graphen beachten. Entsprechend den Tabellen 4.5 und 4.6 auf der Seite 107 ergeben sich für die Kanten die in Tabelle 5.5 dargestellten Attribute.

Werden nun anschließend die inkorrekten Planungsunterlagen des AVA-Systems aktualisiert, so ergibt sich für den Graphen der in Abbildung 5.22 dargestellte Zustand, wenn:

- Die Aktualisierung des Dokumentes 'Wandmengen' intern im Integrationsmodul erfolgte, d.h. die Mengenänderung der Wand wird unter Zuhilfenahme der bereitgestellten Funktionalität (siehe Szenario 5.4.1) durchgeführt.

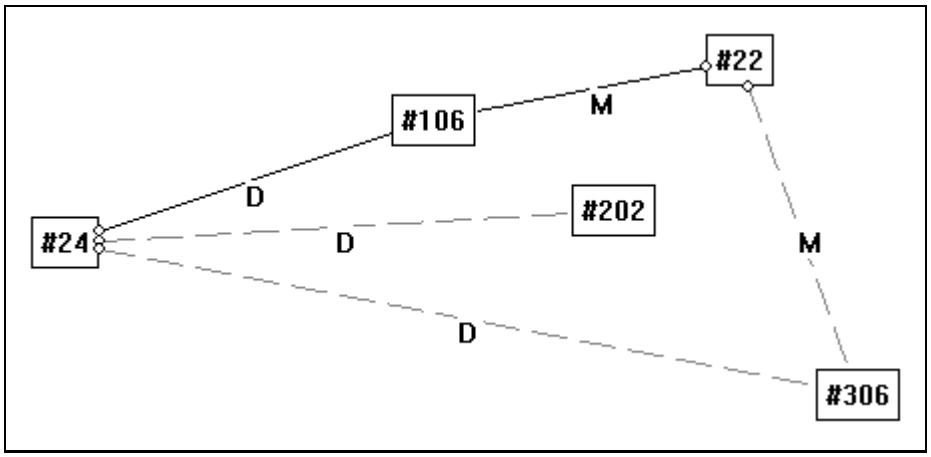


Abbildung 5.22: Zustand des Graphen nach dem Aktualisieren aller Planungsunterlagen

- Die Aktualisierung des Dokumentes 'Mengen von Türen, Fenster und Stürze' im AVA-System erfolgt ist, d.h. durch die Anwendung der vom AVA-System bereitgestellten Funtionalitäten.

Um nun den Zustand *stark konsistent* für die aktualisierten Dokumente zu erreichen, muß der Anwender im Integrationsmodul den Zustand der Kanten auf *beachtet* setzen, deren Endknoten die beteiligten Objekte in den aktualisierten Dokumenten darstellen und für diese Objekte die aufgetretene Veränderung der durch den Startknoten verkörperten Objekte beachten. In diesem Fall sind das die Kanten #24 → #202, #24 → #306 und #22 → #306.

Die anschließende Aktion, die automatisch nach dem Setzen des Kantenattributes λ aufgerufen wird, zerlegt den Graphen $\mathcal{G}$ in seine zusammenhängenden Komponenten und entfernt alle Teilbäume, deren Knoten den Zustand *aktualisiert* und deren Kante den Zustand *beachtet* besitzen. Da der im Beispiel enthaltene Graph zusammenhängend ist und die aufgeführte Bedingung von den im Graphen enthaltenen Knoten und Kanten erfüllt wird, wird die Instanz gelöscht.

Demnach können den Dokumenten keine Objekte mehr zugeordnet werden, die aktualisiert werden müssen. Für alle Planungsunterlagen wird daraus der Zustand *stark konsistent* zugeordnet. Das heißt für das Gesamtmodell, daß alle Dokumente zueinander konsistent sind.

Anhang

Anhang A: Verwendete Begriffe

Anwender:
Beschreibt eine Person, die mittels des Integrationsmoduls Integrationsaufgaben durchführt.

Bearbeiter:
Beschreibt eine Person, die mittels einer externen Applikation ein Modell erzeugt.

Applikation:
Ein Softwareprodukt, das für die Bearbeitung eines Dokumentes bestimmte Funktionalitäten bereitstellt. Die Funktionalitäten sind den zugrundeliegenden Datenstrukturen des Modellierungskonzeptes angepaßt, und erlauben die Darstellung des Datenraums in einer fachspezifischen Notation. Die Bearbeitung des Dokumentes entspricht dem zugrundeliegenden Modellierungskonzept.

externe Applikation:
Ein Softwareprodukt für die Bearbeitung eines Dokumentes, welches die Planung einer bestimmten Bauleistung erlaubt. Externe Applikationen stellen dem Bearbeiter Funktionalitäten zur Verfügung, um eine bestimmte Planungsaufgabe zu realisieren.

Integrationsmodul:
Ein Container für die Bearbeitung der an der Planung beteiligten Dokumente. Das Integrationsmodul stellt dem Anwender Funktionalitäten zur Verfügung, um bestimmte Integrationsaufgaben zu realisieren.

Modellname:
Ein Modellname identifiziert eineindeutig ein Modell im Integrationsmodul. Da das Modell und das Dokument einer Planungsunterlage auf dem gleichen Datenraum aufbauen ist die Verwendung zweier Bezeichner (Modellname, Dokumentname) für die Identifizierung eines Dokuments oder eines Modells hinfällig. Für die Identifizierung eines Dokumentes wie auch eines Modells wird durchgängig der Modellname verwendet.

Information:
Informationen sind immer an einen Informationsträger gebunden. Der Informationsträger codiert die enthaltene Information durch die Annahme eines der Information entsprechenden Zustandes. Durch die Interpretation des Zustandes eines Informationsträgers kann erneut auf die Information zugegriffen werden. Der kleinste atomare Informationsträger wird Datum genannt.

Objektbeschreibung:
Ein gültiger Datentyp, der als Vorlage für ein Objekt verwendet wird. Der Datentyp enthält

mindestens ein Attribut für die persistente Identifizierung der einzelnen Objekte in den einzelnen Dokumenten (eineindeutig).

Typbeschreibung:
Ein gültiger Datentyp, der als Vorlage eine Datenstruktur definiert. Die Datenstruktur beinhaltet keine Attribute für die persistente Identifizierung der Instanz.

Datentyp:
Zusammenfassung von Wertebereich und Operationen zu einer Einheit. Die Menge aller Datentypen vereinigt die Menge aller Objektbeschreibungen und die Menge aller Typbeschreibungen. Jeder Datentyp läßt sich in die Gruppe der atomaren Datentypen und in die Gruppe der zusammengesetzten Datentypen einordnen. Die Menge der gültigen Datentypen liegt zum Zeitpunkt der Verwendung einer Applikation fest.

atomare Datentypen:
Bilden die kleinste Einheit. Die Menge der atomaren Datentypen wird festgelegt.

zusammengesetzte Datentypen:
Erweitern das Datentypsystem einer Applikation durch eine rekursive Verknüpfung bereits bekannter Datentypen. Die in dieser Arbeit unterstützten Verknüpfungen sind Tupel-, Listen- und Mengenkonstruktion sowie die Modellierung von Beziehungen.

Metaklassen:
Bilden im Integrationsmodul das verwendete Datentypsystem. Für die atomaren Datentypen und für die Verknüpfungen von Datentypen existieren im Integrationsmodul jeweils entsprechende Repräsentanten.

Klasseninstanz:
Bezeichnet die Objektbeschreibung eines Objektes im Integrationsmodul (aufbauend auf den Metaklassen).

Objektinstanz:
Bezeichnet ein Objekt im Integrationsmodul aufbauend auf einer Klasseninstanz.

Modell:
Ein Modell beschreibt eine abstrakte Nachbildung der objektiven Realität. Der hierfür verwendete Abstraktionsgrad ist vom Zweck abhängig, für welchen das Modell erzeugt werden soll. Erfahrungen und Fertigkeiten haben die Darstellung eines Modells optimiert sowie ein Modellierungskonzept verfeinert, indem die unbedingt notwendigen Funktionalitäten und Informationen enthalten sind. Bearbeiter bekommen bei der Verwendung einer externen Applikation ihr spezifisches Modell in der Benutzeroberfläche dargestellt.

Modellierungskonzept:
Das Modellierungskonzept legt Richtlinien für die Erzeugung eines Modelles fest. Es beinhaltet die notwendigen Informationen des Modells, die notwendigen Funktionalitäten um ein Modell zu erzeugen und zu bearbeiten sowie die anerkannten Normen und Verfahren für die Bearbeitung des Modells.

Dokument:
Definiert die digitale Darstellung einer Planungsunterlage.

Planungsunterlage:
Bezeichnet die Zusammenfassung von Dokument und Modell.

Instanz:
Adressierter Speicherplatz im Hauptspeicher. Die Größe des verwendeten Speicherplatzes ist vom zugrundeliegenden Datentyp abhängig.

Zustand:
repräsentiert die aktuelle Wertbelegung einer Instanz.

Anhang B: Verwendete Symbole

U - Eine Menge von Umschreibungen. Ein Element $u \in U$ heißt Bezeichner.

S - Eine Menge von gültigen Datentypen. Ein Element $s \in S$ heißt Datentyp.

$\mathcal{K}$ - Die Menge der möglichen Verknüpfungen, um komplexere Datentypen zu konstruieren. Im Integrationsmodul werden folgende Verknüpfungen unterstützt:

- $\mathcal{K}_T$ - Konstruktion eines Tupeltyps
- $\mathcal{K}_L$ - Konstruktion eines Listentyps
- $\mathcal{K}_M$ - Konstruktion eines Mengentyps
- $\mathcal{K}_{REF}$ - Konstruktion eines Beziehungtyps.

$dom(s)$ - Die Funktion liefert den gültigen Wertebereich des entsprechenden Datentyps s zurück.

$z(s)$ - Die Funktion liefert den Zustand einer zugehörigen Instanz zurück. Der Wert ist Element des zum Datentyp gehörenden Wertebereichs.

a - Bezeichnet ein Attribut. Ein Attribut wird mittels eines Tripels definiert, bestehend aus Attributname, Datentyp und einem Schalter, der die möglichen Zugriffe auf den Attributwert steuert.

A - Menge aller Attribute.

W^* - Mengensystem aller Wertebereiche.

σ - Identifikation eines Objektes.

o - Ein Objekt des Integrationsmoduls. Ein Objekt setzt sich aus einem Tripel, bestehend aus einer Objektidentifikation, einer Objektbeschreibung und einem Zustand zusammen.

$\eta(o)$ - Funktion liefert die für dieses Objekt aufgerufene Objektidentifikation zurück.

o_e - Stellvertreterobjekt einer Planungsunterlage im Integrationsmodul.

$\wp$ - Bezeichnet einen Auswahlsatz, eine Menge von Objekten mit gleicher Objektbeschreibung

$\overset{s}{\Theta}$ - Gültige Vergleichsoperationen bezüglich des angegebenen Datentyps s.

$\overset{s}{\Psi}$ - Gültige Zuweisungsoperationen bezüglich des angegebenen Datentyps s.

$\overset{s}{\Omega}$ - Gültige mathematische und logische Operationen bezüglich des angegebenen Datentyps s.

$\overset{T}{\bigcup}$ - Vereinigung zweier typgleicher Datentypen.

$\bigcap^{\tau}$ - Differenz zweier typgleicher Datentypen.

$\setminus^{\tau}$ - Durchschnitt zweier typgleicher Datentypen.

$\stackrel{\tau}{=}$ - Test auf Gleichheit zweier Datentypen.

$\subset^{\tau}$ - Test auf Enthaltensein eines Datentyps in einem anderen Datentyp.

$\mathcal{P}_k(\wp)$ - Projiziert die Objektbeschreibung der im angegebenen Auswahlsatz enthaltenen Objekte auf den angegebenen Datentyp k.

$\mathcal{R}_{u \to u'}(\wp)$ - Benennt den angegebenen Bezeichner u der Objektbeschreibung (der im angegebenen Auswahlsatz enthaltenen Objekte) in den angegebenen Bezeichner u' um.

$\mathcal{J}(\wp_1, \wp_2)$ - Verbund zweier Auswahlsätze.

$\mathcal{X}_{<u,s>}(\wp)$ - Erweiterung des Datentyps um das angegebene Attribut, der den Objekten im angegebenen Auswahlsatz zugrunde liegt.

$\mathcal{S}_B(\wp)$ - Auswahl aller Objekte des angegebenen Auswahlsatzes, die die Bedingung B erfüllen.

$\mathcal{A}(\wp_1, \wp_2)$ - Durchschnitt zweier Auswahlsätze.

$\mathcal{M}(\wp_1, \wp_2)$ - Differenz zweier Auswahlsätze.

$\mathcal{U}_{u\overset{s,u}{\Psi}w}(\wp)$ - Aktualisierung eines Attributes aller Objekte des angegebenen Auswahlsatzes.

$\mathcal{N}(\wp_1, \wp_2)$ - Vereinigung zweier Auswahlsätze.

Σ - Bezeichnet die Menge der übernommenen Objekte einer Planungsunterlage.

O^D - Menge von Objektidentifikationen, deren Objekte nach einer Überarbeitung gelöscht wurden.

O^I - Menge von Objektidentifikationen, deren Objekte nach einer Überarbeitung neu eingefügt wurden.

O^M - Menge von Objektidentifikationen, deren Objekte nach einer Überarbeitung modifiziert wurden.

α^D - Menge von Objektidentifikationen, deren Objekte überprüft werden müssen, da verwendete Objekte gelöscht wurden.

α^M - Menge von Objektidentifikationen, deren Objekte überprüft werden müssen, da verwendete Objekte modifiziert wurden.

ρ - Eine Relation, die zwei Objekte mittels deren Objektidentifikationen in Beziehung setzt.

$\mathcal{G}$ - Graph für die Konsistenzkontrolle der Dokumente.

$\mathcal{V}$ - Menge aller Knoten des Graphen $\mathcal{G}$.

$\mathcal{E}$ - Menge aller Kanten des Graphen $\mathcal{G}$.

ζ - Objektzustand des im Graphen enthaltenen Objektes $\{aktualisiert, aktualisierbar, warten\}$.

λ - Kantenzustand zweier im Graphen enthaltener Objekte $\{beachtet, unbeachtet\}$.

κ - Art der Veränderung des im Graphen enthaltenen Objektes
{ *eingefügt, gelöscht, modifiziert, unverändert, unbekannt* }.

Die noch verbleibenden mathematischen Symbole werden gemäß der in [Gell77] aufgeführten Definitionen verwendet.

Anhang C: Eidesstattliche Erklärung

Ich erkläre hiermit in Kenntnis der strafbaren Folgen einer eidesstattliche Falschaussage an Eides Statt, daß ich die vorliegende Arbeit ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen direkt oder indirekt übernommenen Daten, Methoden und Konzepte sind unter Angabe der Quellen gekennzeichnet.

Weimar, 30.09.1999

Anhang D: Lebenslauf

Geburtstag und -ort	29.09.1966 in Borna, Sachsen
Personaldaten	verheiratet, ein Kind
Berufsbildung	• 1973-1983 zehnklassige allgemeinbildende polytechnische Oberschule Altenburg • 1983-1985 Ausbildung zum Facharbeiter Berufsbezeichnung: Maurer • 1984-1986 Einrichtung der Erwachsenenbildung Altenburg Abschluß: Abitur • 1988-1994 Bauhaus-Universität Weimar Abschluß: Dipl.-Informatiker Thema der Diplomarbeit: Schnittstellengestaltung zwischen CAD- und AVA-Systemen
Wehrdienst	1986-1988
Berufstätigkeit	• 1985-1986 Tätigkeit als Maurer PGH Bau Altenburg, Altenburg • 1994-heute wissentschaftlicher Mitarbeiter am Lehrstuhl Informatik im Bauwesen Bauhaus-Universität Weimar

Veröffentlichungen

- Schneider U.
  Schnittstellengestaltung zwischen CAD- und AVA-Systeme
  Diplomarbeit am Lehrstuhl Informatik im Bauwesen
  Bauhaus-Universität Weimar, 1994
- Schneider U.
  Bearbeitung fachübergreifender Daten
  VDI Fortschrittsberichte Reihe 4 Nr. 135
  8. Forum Bauinformatik Cottbus 1996
- Schneider U.
  Standardisierug der Kommunikation als Integrationsansatz für das Bauwesen
  VDI Fortschrittsberichte Reihe 4 Nr. 140
  9. Forum Bauinformatik Dresden 1997
- Schneider U., Beucke, K.
  Standardisierug der Kommunikation als Integrationsansatz für das Bauwesen
  Internationales Kolloquium über Anwendungen der Informatik und Mathematik in Architektur und Bauwesen - IKM, Weimar 1997
- Beucke, Schneider
  Active Integration Concepts Based on a Communikation Standard
  Structures for the Future - The Search for Quality
  IABSE Symposium Rio de Janeiro 1999
  Session 9, Information Technology
- Schneider U.
  Standardisierug der Kommunikation als Integrationsansatz für das Bauwesen
  VDI Fortschrittsberichte Reihe 4 Nr. 140
  10. Forum Bauinformatik Darmstadt 1999

6 Ausblick

In der vorliegenden Arbeit werden die Grundkonzepte eines Integrationsansatzes vorgestellt, der auf einer aktiven Integration der einzelnen Anwender (Fachplaner) beruht. Der Integrationsansatz verfolgt das Ziel, eine Aufgabenteilung einzuführen, in der die einzelnen Anwender und die zu entwickelnde Software optimal bezüglich ihrer wissensverarbeitenden Stärken eingesetzt werden, um somit die entsprechenden Vorteile eines Integrationsansatzes ohne Beteiligung von Integrationssoftware mit denen eines Integrationsansatzes mit Integrationssoftware zu vereinen.

Aufgabe des Anwenders ist es, mittels bereitgestellter Aktionen die Integrationsaufgaben zu realisieren, wobei der Anwender die Semantik der beteiligten Objektbeschreibungen analysieren muß. Die Software beschränkt sich auf die Bereitstellung der notwendigen Aktionen, auf die Speicherung umfangreicher Informationsinhalte und auf das Protokollieren von Veränderungen, wobei diese Funktionalitäten unabhängig von der Semantik konkreter Objekte sind.

Die bezüglich der Objekte semantikfreie Spezifikation der Funktionalitäten führt zu einer Standardisierung der Kommunikation für folgende Einsatzgebiete:

- Kommunikation externer Applikationen mit dem Integrationsmodul über COM-Schnittstellen für das Einbetten eines Dokumentes in das Integrationsmodul (OLE) und für den Zugriff auf die jeweiligen Objektbeschreibungen und -zustände in den betrachteten Dokumenten via ActiveX-Automation. Weitere COM-Schnittstellen, die speziell für die Kommunikation zwischen den einzelnen Applikationen und dem Integrationsmodul entwickelt wurden, ermöglichen den Zugriff auf die Objekte eines Dokumentes und das Erzeugen der Objekte in einem angegebenen Dokument.

- Bereitstellen eines Interpreters für die Manipulation von Objektbeschreibungen und -zuständen, der individuell erstellte Kommandos interpretiert, die zur Laufzeit des Integrationsmoduls erzeugt werden können.

Voraussetzung für die Standardisierung der Kommunikation ist eine auf einem Metamodell basierende Implementierung der im Integrationsmodul verwendeten Objekte.

Der Einsatz des Integrationsansatzes ist jedoch von der Robustheit der Implementierung abhängig, die maßgeblich von der Integrität zwischen den im Integrationsmodul verwendeten Objekten und den in den Dokumenten eingebetteten Objekten bestimmt wird. Eine weitere Voraussetzung bezüglich der Robustheit, die bereits erfüllt wurde, ist die Beschreibung der verwendeten Aktionen mittels einer formalen Semantik, die unabhängig von der Semantik der beteiligten Objekte ist.

Für das Sicherstellen der Robustheit im Integrationsmodul existiert noch Entwicklungsbedarf. Die Integrität zwischen den eingebetteten Dokumenten und den erzeugten Datenstrukturen im Integrationsmodul ist nach bestimmten Aktionen nicht gewährleistet, z.B. der Anwender übernimmt Objekte in ein Dokument, aktualisiert jedoch dessen Referenz im Integrationsmodul nicht. Ein in Frage kommendes Konzept hierfür sind geschachtelte Transaktionen. Wie weit diese für die Sicherung der Robustheit eingesetzt werden können, muß noch untersucht werden.

Der uneingeschränkte Zugriff auf den Datenraum des Integrationsmoduls für jeden Anwender ist aus Gründen der Haftung nicht zulässig. Die Verwaltung von Rechten bedingt eine Erweiterung des Integrationsmoduls um ein Benutzermanagement. Entscheidend für die Spezifikation des Benutzermanagements sind die notwendige Tiefe, in der das Benutzermanagement noch regelnd eingreift, und welche Rechte für die Aufgaben der Integration eine dominante Rolle spielen.

Eine weitere Untersuchung, welche in dieser Arbeit noch nicht durchgeführt wurde, ist die Analyse des Einsatzes des Integrationsmoduls. Die Analyse muß Aufschluß über die Anwenderakzeptanz, über die Effizienz und den Nutzen geben. Außerdem muß noch geklärt werden, ob die Darstellungsvarianten der Objektbeschreibungen und -zustände im Integrationsmodul für die Anwender ausreichen oder ob weitere Darstellungsmöglichkeiten entwickelt werden müssen.

Des weiteren lassen sich Verbesserungen angeben, die die Effizienz des Integrationsansatzes erhöhen. Diese Verbesserungen lassen sich den beiden Kategorien 'Erweiterungen der Anwenderschnittstelle' (Abschnitt 6.1) und 'konzeptionelle Erweiterungen' (Abschnitt 6.2) zuordnen, die in den folgenden beiden Abschnitten vorgestellt werden.

Die Entwicklung einer Integrationskomponente, welche den Integrationsprozeß interaktiv gestaltet, erlaubt eine anwenderfreundliche Handhabung des Integrationsprozesses. Dabei erlaubt dieses Integrationswerkzeug nicht nur die Verknüpfung der technischen Bereiche des Planungsprozesses untereinander, sondern ermöglicht des weiteren auch die Integration der wirtschaftlichen und bauausführenden Bereiche. Demnach unterstützt dieses Integrationswerkzeug die Verknüpfung von geometrischen und topologischen Informationen, funktionalen Aspekten sowie Zeitangaben (Termine) und anfallenden Kosten.

Die Möglichkeit, alle anfallenden Informationen in einem Gesamtmodell konsistent zu verwalten, ermöglicht letztendlich, die anfallenden Tätigkeiten auf einer Baustelle bereits im Rechner zu simulieren. Mit Hilfe solch einer Simulation lassen sich bereits Planungsfehler erkennen, die normalerweise erst auf der Baustelle erkannt werden, d.h. wenn es schon zu spät für eine durchgängige Fehlerbehebung ist. Erst durch den Einsatz der Simulation ist nach Meinung des Verfassers ein Qualitätssprung in der Sicherung der Qualität der Planungsunterlagen möglich.

6.1 Erweiterungen der Anwenderschnittstelle

6.1.1 Erweiterungen der Kommandosprache

Im Abschnitt 4.2.3 auf Seite 85 wurde die Syntax der verwendbaren Kommandos für die Manipulation der Objektzustände und deren Objektbeschreibungen spezifiziert. Die Implementierung des zu diesen Kommandos gehörenden Interpreters legt die verwendete Art des Objektvergleiches sowie der Objektkopie in den jeweiligen Kommandos fest.

Für das Vergleichen sowie für das Kopieren von Objekten existieren jedoch verschiedene Arten, je nachdem ob für den Vergleich oder für das Kopieren aggregierte oder assoziierte Instanzen verwendet werden oder nicht. Der Objektvergleich kann noch einmal um die Betrachtung der beteiligten Objektidentifikatoren erweitert werden.

Demnach sind für den Objektvergleich folgende Arten signifikant:

- flache Gleichheit: Objektvergleich ohne Objektidentifikatoren und ohne Einbeziehung aggregierter oder assoziierter Instanzen,
- tiefe Gleichheit: Objektvergleich ohne Objektidentifikatoren und mit Einbeziehung aggregierter oder assoziierter Instanzen,
- identische Gleichheit: Objektvergleich mit Objektidentifikatoren und mit Einbeziehung aggregierter oder assoziierter Instanzen.

Ähnliches läßt sich für das Kopieren angeben:

- flaches Kopieren: Erzeugen einer Objektkopie, wobei die Kopie dieselben aggregierten oder assoziierten Instanzen verwendet wie das Original,
- tiefes Kopieren: Erzeugen der Objektkopie des Originals sowie Kopien für alle aggregierten oder assoziierten Instanzen.

Da jedes Objekt über eine eineindeutige Identifikation verfügt, muß bei der Objekterzeugung jedem Objekt ein neuer Identifikator zugewiesen werden. Demnach kann das Kopieren keine Objekte mit identischen Objektidentifikatoren erzeugen.

Der zur Zeit entwickelte Interpreter verwendet ausschließlich die flache Gleichheit und das flache Kopieren. Mit dem Erweitern der Kommandos um eine Optionsliste, deren Optionen die Art des Objektvergleiches und der Objektkopie angeben und die je nach Kommando variieren, kann der Interpreter über die obigen Aspekte verfügen und entsprechend gesteuert werden.

Die Verwendung von Optionen für das Vergleichen und für das Kopieren von Objekten erlaubt eine bessere Möglichkeit mittels der Kommandos die Aufgaben der Integration zu realisieren, indem den Anwendern eine gezieltere Auswertung zur Verfügung gestellt wird. Die Verwendung der identischen Gleichheit bedingt jedoch, daß die Erzeugung der Objektidentifikatoren im dynamischen Datenraum von der Kommandosprache durch die Angabe einer entsprechenden Option unterstützt wird und von den Anwendern gesteuert werden muß.

6.1.2 Erweitern der dynamischen Funktionsschnittstelle

Die Einsatzgebiete der Funktionsschnittstelle lassen sich in zwei Gruppen einteilen, je nachdem ob eine Funktion oder eine Methode der Schnittstelle eingesetzt werden soll. Die wesentliche Aufgabe von Funktionen ist es, aus übergebenen Attributen eines aktuellen Objektes oder Konstanten einen resultierenden Wert abzuleiten, der über den Funktionsnamen zurückgegeben wird. Methoden operieren im Gegensatz zu Funktionen auf Objektmengen. Die Aufgabe einer Methode ist die Berechnung von Attributwerten aus der - als zweiten Parameter - angegebenen Objektmenge, die einem - als ersten Parameter - übergebenem Objekt zugewiesen werden. Zusätzlich muß die Methode dem Integrationsmodul die entsprechenden Abhängigkeiten zwischen den einzelnen Objekten mitteilen.

Ein typisches Einsatzgebiet für die Funktionen ist beispielsweise die Ermittlung eines Abstandes aus Punkten oder die Berechnung des Flächeninhaltes eines geschlossenen Polygonzuges. Methoden können beispielsweise für die Behandlung von Beziehungen eingesetzt werden, die nicht

direkt durch entsprechende Attributwerte verfügbar sind, wie die Zuordnung von Rohren zu den Wänden, die von den Rohren durchdrungen werden, oder die Zuordnung von Raumnummern zu den Wänden, die diesen Raum bilden.

Zur Zeit unterstützt das Integrationskonzept nur das Einbringen neuer Funktionen mittels der OLE-Automation. Hierzu sind folgende Arbeitsschritte notwendig:

- Generieren einer neuen oder die Erweiterung einer bestehenden Funktionsschnittstelle, basierend auf der OLE-Automation.
- Implementieren der gewünschten Funktionen. Die Parameterübergabe muß auf den Automation-Schnittstellen der im Integrationsmodul implementierten Metaklassen aufbauen.
- Registrieren der Klasse im Betriebssystem.
- Erweitern der im Integrationsmodul zu benutzenden Klassenverweise via OLE-Automation durch das Markieren der Funktionsschnittstelle.

Nach diesen Schritten kann diese Funktionsschnittstelle mit ihren einzelnen Funktionen im Integrationsmodul innerhalb einer UPDATE-Anweisung verwendet werden.

Für das Verwenden einer Methodenschnittstellenklasse sind ähnliche Arbeitsschritte notwendig. Forschungsbedarf besteht hierbei jedoch bezüglich der Spezifikation der Signatur solch einer Methode und wie diese Methoden in das Integrationsmodul integriert werden können.

6.1.3 Persistente Speicherung der Kommandos (Makros)

Im Abschnitt 5 wurden einige Kommandos für ausgewählte Integrationsaufgaben vorgestellt. Vorstellbar ist es demnach, daß im praktischen Einsatz eine Unmenge von Kommandos benötigt werden, die aus Gründen der Wiederverwendung persistent im Integrationsmodul vorliegen müssen.

Die persistente Speicherung vereinfacht die Handhabung des Integrationsmoduls durch das Ermöglichen einer Teilautomatisierung des Integrationsprozesses. Die Speicherung der Kommandos muß die Eineindeutigkeit der Kommandos gewährleisten, die von folgenden Kriterien abhängt:

- Speicherung nutzerunabhängiger Kommandos, welche global allen Anwendern zur Verfügung stehen,
- Speicherung nutzerabhängiger Kommandos, welche nur dem Eigentümer zur Verfügung stehen,
- zusammenfassen mehrerer Kommandos zu einem nutzerunabhängigen Makro und
- zusammenfassen mehrerer Kommandos zu einem nutzerabhängigen Makro.

Die Zusammenfassung mehrerer Kommandos zu einem Makro kann aus Kommandos oder bereits existierenden Makros bestehen, die sowohl nutzerabhängig als auch nutzerunabhängig sein können.

Problematisch ist hierbei die eindeutige Identifizierung eines Kommandos sowie die Gewährleistung, daß die erzeugten Objekte eines Kommandos oder Makros in einem eindeutigen Auswahlsatz zusammengefaßt werden. Demnach bedarf dieser Punkt weiterer Untersuchungen, um eine Ordnungsstruktur für alle Kommandos und Makros zu entwickeln. Ausgehend von der Ordnungsstruktur muß eine eindeutige Identifizierung ableitbar sein. Die Ordnungsstruktur muß demnach die obigen Aspekte beachten und auf den unterschiedlichen Applikationen und Anwenderanforderungen aufbauen.

Außerdem muß für jedes globale Kommando oder Makro eine Dokumentation verfügbar sein, so daß ein Fremder das Kommando interpretieren und anschließend verwenden kann. Die Dokumentation muß den Inhalt des Kommandos (Makros) wiedergeben sowie deren Handhabung beschreiben. Das Auffinden eines bestehenden, jedoch für einen bestimmten Anwender noch unbekannten Kommandos, kann nur durch das Durcharbeiten der einzelnen Dokumentationen erfolgen. Demnach müssen geeignete Suchstrategien innerhalb der resultierenden Dokumentation aller verwendbaren Makros und Kommandos entwickelt werden, wenn das Auffinden eines verfügbaren Kommandos effizient erfolgen soll.

Eine weitere Aufgabe, die Gegenstand der persistenten Speicherung ist, ist die Übernahme bestehender nutzerabhängiger Kommandos (Makros) aus einem anderen Projekt in das aktuelle. Hierfür muß eine weitere Methodik angegeben werden, die den Umgang mit aggregierten Kommandos und Makros festlegt.

Denkbar für die Spezifikation der Ordnungsstruktur ist ein auf das Benutzermanagement ausgerichteter Namensbereich sowie das Einführen von Sichbarkeitskriterien. Demnach ist jedes Kommando durch den Zugriff *Namensbereich::Name* eindeutig identifizierbar, wenn für die Erzeugung eines neuen Namensbereiches mindestens das aktuelle Projekt, der Eigentümer und die beteiligten Applikationen beachtet werden und wenn *Name* den hierarchischen Komponentenzugriff auf das bestimmte Makro (Kommando) angibt. Wie tragbar dieser Ansatz ist, muß noch nachgewiesen werden.

6.2 Konzeptionelle Erweiterungen

6.2.1 Behandlung inverser Beziehungen

Im Abschnitt 4.2.5 wurde das Konzept für die Wahrung der Konsistenz erläutert. Das Konzept basiert auf dem Prinzip der Vorlageberechtigung. Demnach gibt es im Bauwesen Richtlinien, welche Information von welchem Fachplaner festgelegt wird und wer diese Information verwenden muß.

Ein Integrationsansatz, der die Belange der Anwender unterstützt, sollte die existierenden Arbeitsvorgänge beachten. Speziell für die Vorlageberechtigungen trifft diese Aussage nicht im gewünschten Ausmaß zu. Gegenwärtig dominieren Kompromißlösungen, deren Erarbeitung die Richtlinien für die Vorlageberechtigung außer Kraft setzen, d.h. es wird die günstigste Lösung für den Bauherrn gesucht, indem alle beteiligten Fachplaner Vorschläge einbringen.

Die Variante, daß ein anderer Fachplaner eine Information festlegt, die anschließend von dem als vorlageberechtigt eingesetzter Fachplaner verwendet wird, muß demnach vom Integrationsansatz

unterstützt werden. Diese Unterstützung verändert nicht die Richtung der Abhängigkeitsbeziehung, die entsprechend der Richtlinien für die Vorlageberechtigung angegeben werden soll, sondern negiert die Verwendung der Beziehung.

Zur Zeit wird im Integrationsmodul nur die Behandlung der Beziehungen entsprechend der durch die Vorlageberechtigung definierten Reihenfolge unterstützt. Zusätzlich können bestimmte Relationen ausgewählt werden, um diese als *invers* zu kennzeichnen. Eine globale Nachricht wird erzeugt, sobald eine invers definierte Abhängigkeit vom Integrationsmodul bearbeitet wird.

Notwendig ist es jedoch, mindestens die gleichen Möglichkeiten für die Kontrolle und Überwachung von Veränderungen invers definierter Beziehungen softwaretechnisch zu unterstützen. Die Spezifikation des Konzeptes für die Behandlung von inversen Beziehungen muß für den vorlageberechtigten Fachplaner die Kontrolle von Veränderungen in verwendeten Dokumenten erlauben und dokumentieren. Demnach wird für die Behandlung der inversen Beziehungen ein weiterer Graph benötigt. Problematisch hierbei ist es, die beiden Graphen konform zu halten.

Unterbreitet ein nicht vorlageberechtigter Fachplaner einem vorlageberechtigten Fachplaner einen Vorschlag, so werden diese Veränderungen in den Graphen für die Behandlung der inversen Beziehungen eingetragen. Der vorlageberechtigte Fachplaner kann die Veränderungen analysieren und gegebenenfalls übernehmen. Hierzu stellt das Integrationsmodul ausreichende Funktionalitäten zur Verfügung. Nach dem Aktualisieren des vom vorlageberechtigten Fachplaner bearbeiteten Dokumentes erfolgt das Erkennen der Veränderungen im Integrationsmodul und somit die Aktualisierung des Graphen $\mathcal{G}$ in üblicher und bereits vorgestellter Weise.

6.2.2 Verwenden von Containerklassen

Containerklassen verwalten eine beliebige Anzahl typgleicher Objekte und stellen für diese allgemeingültige Verwaltungsfunktionalitäten wie Einfügen eines Objektes, Zugriff auf ein Objekt sowie Löschen eines Objektes zur Verfügung. Gewöhnlich erlauben Containerklassen einen sequentiellen Zugriff auf jedes in der Containerklasse enthaltene Objekt. Semantisch entsprechen diese Verwaltungseinheiten einer allgemeingültigen Liste (Array), sobald die Anzahl der Objekte festliegt und sobald nur iterativ auf die Objekte zugegriffen wird.

Zur Zeit wird im Integrationsmodul auf die Unterstützung von Containerklassen verzichtet. Grund hierfür ist, daß für den iterativen Zugriff auf die Objekte einer Containerklasse keine Standardisierungen existieren, wobei die dafür notwendigen Algorithmen überschaubar sind. Der Verzicht auf Containerklassen bedingt jedoch nicht, daß prinzipiell auf die Übernahme von Beziehungen, die mittels Containerklassen in den jeweiligen Dokumenten vorliegen, verzichtet werden muß.

Zum einen kann mittels einer Voraussetzung an die zu integrierenden Applikationen gefordert werden, daß die Attribute, die auf einer Containerklasse aufbauen, nicht die Automation-Schnittstelle der Containerklassen veröffentlichen, sondern die enthaltenen Objekte einer Containerklasse in einem Feld zusammenfassen und dieses Feld als Attributwert dem Automation-Client zur Verfügung stellen. Zum anderen ermöglicht das dynamische Funktionskonzept die Auswertung dieser Beziehungen durch entsprechende, dynamisch eingebundene Methoden.

Da jedoch die Automation-Schnittstelle nicht vorrangig für die Integration entwickelt wird, sondern eine Möglichkeit für den Bearbeiter darstellt, eine Applikation nachträglich zu konfigurieren, ist der Verzicht auf Containerklassen in der Automation-Schnittstelle nicht akzeptabel. Die Verwendung des Funktionskonzeptes beeinträchtigt die Überprüfung der Objekte, da aufgetretene Veränderungen bezüglich ihrer modellierten Beziehungen nicht vom Prozeß 'Wahrung der Konsistenz' erkannt werden können. Deswegen muß langfristig gesehen die Verwendung von Containerklassen im Integrationsmodul unterstützt werden.

Das letztendliche Ziel, das mit der Übernahme von Containerklassen im Integrationsmodul verfolgt wird, ist, daß ein zu einem Objekt im Integrationsmodul gehörendes Attribut alle Objekte referenziert, die Elemente der zugrundeliegenden Instanz einer Containerklasse sind. Hierfür muß jedoch ein Algorithmus angegeben werden, der je nach Containerklasse unterschiedliche Datentypen und Methodensignaturen verwendet.

Ein möglicher Lösungsansatz ergibt sich durch das Bereitstellen von verschiedenen Umgebungen, die verschiedene Algorithmen für den sequentiellen Zugriff auf die einzelnen Elemente von Containerklassen implementieren. Jede Umgebung verwendet hierfür abstrakte Methodennamen und Variablen mit abstrakten Datentypen. Wird durch den Anwender eine Aktion gestartet, die die Übernahme aller referenzierten Objekte eines auf einer Containerklasse basierenden Attributes beinhaltet, dann erfolgt die Auswahl einer entsprechenden Umgebung und die Zuordnung der in der Schnittstelle vorhandenen Methoden und Datentypen zu den in der Umgebung verwendeten Methoden und Datentypen. Die Implementierung der Aktion benutzt die getroffenen Zuordnungen für den sequentiellen Zugriff auf die einzelnen Elemente und baut die entsprechenden Beziehungen im Integrationsmodul nach.

Noch zu klären in diesem Zusammenhang ist der Umfang der notwendigen Umgebungen sowie die zu diesen Umgebungen gehörenden Zuordnungen von Methodensignaturen und Variablentypen, und wie diese persistent mit den entsprechenden Attributen verbunden werden können.

6.2.3 Erzeugen von Objekten in einem Zieldokument

Gegenstand des Teilprozesses 'Erzeugen von Objekten' ist die Übernahme manipulierter Objekte in ein Zieldokument. Die manipulierten Objekte ergaben sich aus dem Anwenden der im Abschnitt 4.2.3 erläuterten Kommandos, die auf selektierten Objekten in den Quelldokumenten operierten.

Der für die Erzeugung der Objekte im Zieldokument angewendete Algorithmus erzeugt permanent neue Instanzen im Zielmodell. Die Erzeugung umfaßt das Objekt selbst sowie auch alle referenzierten Instanzen des Objektes. Probleme entstehen, falls ein zu erzeugendes Objekt eine Menge von bereits im Zieldokument existierenden Instanzen referenziert. In diesem Fall dürfen die bereits im Zieldokument existierenden Instanzen nicht noch einmal erzeugt werden.

Die implementierte Variante des Prototyps setzt demnach voraus, daß das zu erzeugende Objekt nur Instanzen referenziert, die noch nicht im Zieldokument existieren. Der Anwender muß demnach die referenzierten Instanzen, die bereits im Zieldokument existieren, durch die Anwendung der Kommandos des im Abschnitt 4.2.3 vorgestellten Interpreters entfernen, das Objekt mit den noch nicht vorhandenen referenzierten Instanzen erzeugen und anschließend die persi-

stenten Objektbeziehungen im Integrationsmodul aktualisieren, damit letztendlich das Objekt im Integrationsmodul alle gewünschten Beziehungen enthält.

Eine Vereinfachung hierfür kann sein, ein besonderes Kommando vorzusehen, welches ein zusätzliches Attribut der Metaklasse $\mathcal{K}_{Ref}$ auf einen bestimmten Wert setzt, falls die referenzierte Instanz bereits im zugrunde liegenden Dokument existiert. Der Algorithmus für die Erzeugung der Objekte wertet das zusätzliche Attribut aus und gibt die entsprechende Referenz der Instanz zurück. Wenn die Instanz schon existiert wird deren Referenz ermittelt. Im anderen Fall wird die Instanz erzeugt und die Referenz der neu erzeugten Instanz verwendet.

Die Voraussetzung hierfür ist jedoch, daß der Objekttyp der referenzierten Instanz über die für die Identifizierung der Instanz im Dokument benötigten Attribute verfügt.

Literaturverzeichnis

[Beu93] Beucke K.
Stand der Integration von Statik und Bemessung im Entwurfs- und Konstruktionsprozeß
Fachtagung Baustatik-Baupraxis, Tagungsheft BB5, München 1993

[Beu96] Beucke, K.
CAE - Bauwerksmodelle Sachstandsbericht im Auftrag des Deutschen Betonvereins (DBV - Nr. 188)
Wiesbaden 1996

[BMBF] Bundesministerium für Bildung und Forschung
Arbeitspapier vom 17.03.1997

[Bod95] Bodendiek R., Lang R.
Lehrbuch der Graphentheorie, Band 1
Spektrum Akademischer Verlag Heidelberg/Berlin/Oxford, 1995

[Booch94] Booch G.
Object-Oriented Analysis And Design-with applications
The Benjamin/Cummings Publishing Company,Inc., 1994

[Box98] Box D.
COM-Microsoft Technologie für komponentenbasierte Softwaretechnologie
ADDISON-WESLEY, 1998

[Bret98] Bretschneider D.
Modellierung rechnerunterstützte, kooperativer Arbeit in der Tragwerksplanung
Dissertation Ruhr-Universität Bochum, 1998

[Brock95] Brockschmidt K.
Inside OLE second edition
Microsoft Press,1995

[Burk97] Burkhardt R.
UML-Unified Modeling Language
Objektorientierte Modellierung für die Praxis
ADDISON-WESLAY, 1997

[Cat93] Cattell R.G.
The object database standard: ODMG-93
Objektorientierte Datenbank Management Group
San Mateo, Calif. : Kaufmann, 1993

[Dre89] Dresch P., Frobel G., Koschorreck H.-J.
Informatik für die Sekundarstufe II, Band 2
Algorithmen und Datenstrukturen
Verlag Ferdinand Shöning, Paderborn, 1989

[FAT97] Kolbe P., Pfennigschmidt S., Pahl P. J.
Verteiltes Facility Management in Telekommunikationsnetzen
Abschlußbericht, Technische Universität Berlin, 1997

[Freu98] Freundt M.
Generische Bestimmung von Wegen in Netzen
Diplomarbeit Technische Universität Berlin, 1998

[Gam96] Gamma E., Helm R., Johnson R., Vlissides J.
Entwurfsmuster
Elemente wiederverwendbarer objektorientierter Software
ADDISON-WESLAY, 1996

[Gell77] Gellert W.
Kleine Enzyklopädie Mathematik
VEB Bibliographisches Institut Leipzig, 1977

[Haas] Haas W., Frank A.,
Building Elements Using Explicit Shape Representation
http://www.haspar.de/Ap225/StepByStep_deu.htm

[Hel86] Helmerich, Schwindt
CAD/CAE im Vergleich
VEB Fachbuchverlag Leipzig, 1986

[Heu92] Heuer A.
Objektorientierte Datenbanken, 1.Auflage
ADDISON-WESLEY, 1992

[IAI99] Industry Foundation Classes - Release 2.0
Specifications Volume 1, AEC/FM Processes Supported By IFC, Beta
1999

[Ilie97] Ilieva D., Pahl P. J.
Verteilte Bearbeitung von Zeichnungen
Internationales Kolloquium über Anwendungen der Informatik und Mathematik in Architektur und Bauwesen - IKM
Bauhaus-Universität Weimar, 26.2. - 1.3. 1997
http://www.uni-weimar.de/∼ikm/

[Jun90] Jungnickel D.
Graphen, Netzwerke und Algorithmen
Wissenschaftsverlag Mannheim/Wien/Zürich, 1990

[Knau83] Knaur D.
Knauers etymologisches Lexikon
Lexigrafisches Institut München, 1983

[Kret94] Kretzschmar H. u.a.
Computergestützte Bauplanung
Verlag für Bauwesen, 1994

[Loe86] Loeckx J., Mehlhorn K., Wilhelm R.
Grundlagen der Programmiersprachen
B. G. Teubner Stuttgart, 1986

[Lück96] Lück v. H.
Entwicklung generischer Muster für die Systemarchitektur integrierter Programmsystem im Bauwesen
Dissertation Ruhr-Universität Bochum,1996

[Mant91] Mantscheff J.
Einführung in die Baubetriebslehre Teil 1
Werner Ingenieur-Texte 23, Werner Verlag 1991

[Mack95] Mackert M.
Objektorientierte und wissensbasierte Modellierung für die Statik am Beispiel Tunelbau
Dissertation Technische Universität München, 1995

[Mey90] Meyer B.
Objektorientierte Softwareentwicklung
Hanser-Verlag, 1990

[MIKO97] Thierfelder J., Grosche A., Willenbacher H.
Management- und Informationssystem für den Komplettbau
Abschlußbericht, Bauhaus-Universität Weimar, 1997

[Neu90] Neumark
Normierte Algebren
Verlag der Wissenschaften, Berlin 1990

[OLE 2] Programmer's Referenz
Working with Windows Objects
Microsoft Press,1993

[Pahl] Pahl P. J., Ilieva D.
Eine Datenbahn für das Bauen
http://ifb.bv.tu-berlin.de/DATENBAHN/datenbahn.html

[Roth87] Rothhardt G.
Praxis der Softwareentwicklung
Dr.Alfred Hüthig Verlag Heidelberg, 1987

[Ruep93] Rüppel U.
Objectorientiertes Management von Produktmodellen der Tragwerksplanung
Dissertation Darmstadt 1993.

[Schad94] Schader M.
Objektorientierte Systemanalyse
Springer Verlag, 1994

[Schad97] Schader M.
Objektorientierte Datenbanken
Die C++ Anbindung des ODMG-Standards
Springer Verlag, 1997

[Schnei92] Schneider K. J.
Bautabellen mit Berechnungshinweisen, Beispielen und eropäischen Vorschriften
Werner-Verlag 1992

[Schnei94] Schneider U.
Schnittstellengestaltung zwischen CAD- und AVA-Anwendungen
Diplomarbeit, Bauhaus-Universität Weimar, 1994

[Sed92] Sedgewick R.
Algorithmen
ADDISON-WESLAY,1992

[Shie89] Shields M. W.
Serielle und parallele Automaten
Einführung in die Theorie
VCH Verlagsgesellschaft Weinheim, 1989

[Tan95] Tanenbaum A. S.
Moderne Betriebssysteme, 2. Auflage
Hanser Studienbücher der Informatik, 1995

[Watt96] Watt D. A.
Programmiersprachen : Konzepte und Paradigmen
Hanser-Verlag, 1996

[Zim88] Zima H.
Compilerbau I
Analyse
Wissenschaftsverlag Mannheim/Wien/Zürich, 1988

[Hel86] Helmerich, Schwindt
CAD/CAE im Vergleich
VEB Fachbuchverlag Leipzig, 1986

Danksagung

Während meiner fünfjährigen Arbeit am Lehrstuhl 'Informatik im Bauwesen' an der Bauhaus-Universität Weimar konnte ich mich immer auf die tatkräftige Unterstützung der am Bereich beschäftigten Kollegen verlassen. Ihnen gebührt mein vollster Dank. Namentlich möchte ich mich bei den Kollegen Dr.-Ing. W. Huhnt, Dr.-Ing. H. Kirschke, B. Firmenich, M. Freundt, T. Friedrich, A. Grosche und R. Schumann für ihre Diskussionsbereitschaft bedanken.

Besondern bedanken möchte ich bei meinem Betreuer, Prof. Dr.-Ing. K. Beucke, der mit mir die Begeisterung für die interaktive und iterative Arbeistweise des Anwenders während des Integrationsprozesses teilte und der mich die ganze Zeit großzügig durch Hinweise und Ratschläge unterstützte. Zusätzlich möchte ich mich bei ihm für die Übernahme des Hauptreferats bedanken.

Für die Bereitschaft der Übernahme der Koreferate möchte ich mich bei Herrn Prof. Dr.-Ing. P. J. Pahl und Herrn Prof. Dr. C. Wüthrich bedanken.

Des weiteren möchte ich mich bei den technischen Mitarbeiter M. Bieber und J.-U. Wagner bedanken. Ihre sofortige Hilfe bei den zahlreichen Hard- und Sofwareproblemen trugen maßgeblich zum Gelingen der Arbeit bei. Für die redaktionelle Überarbeitung gebührt C. Diez Dank.

Darüber hinaus möchte ich mich bei meiner Familie bedanken. Ihre Unterstützung und ihre Rücksichtnahme ermöglichten die ungestörte Bearbeitung und unterstützten den Erfolg der Arbeit.